Martínez, M., García, J., Fernández, F. (2018). On-Line case-based policy learning for automated planning in probabilistic environments. International Journal of Information Technology & Decision Making, 17(3), pp. 763-800

DOI: https://doi.org/10.1142/S0219622018500086

International Journal of Information Technology & Decision Making

ON-LINE CASE-BASED POLICY LEARNING FOR AUTOMATED PLANNING IN STOCHASTIC ENVIRONMENTS

MOISÉS MARTÍNEZ

Faculty of Natural and Mathematical Sciences, King's College of London
Strand Campus, Bush House, 30 Aldwych
London, WC2B 4BG, United Kingdom

JAVIER GARCÍA AND FERNANDO FERNÁNDEZ

Computer Science Department, Universidad Carlos III de Madrid
Avenida de la Universidad, 30
Leganés 28911, Madrid, Spain

Many robotic control architectures perform a continuous cycle of sensing, *reasoning* and acting, where that *reasoning* can be carried out in a reactive or deliberative form. Reactive methods are fast and provide the robot with high interaction and response capabilities. Deliberative reasoning is particularly suitable in robotic systems because it employs some form of forward projection (reasoning in depth about goals, preconditions, resources, and timing constraints) and provide the robot reasonable responses in situations unforeseen by the designer. However, this reasoning, typically conducted using Artificial Intelligence techniques like Automated Planning, is no effective for controlling autonomous agents which operate in complex and dynamic environments. Deliberative planning, although feasible in stable situations, takes too long in unexpected or changing situations which require re-planning. Therefore, planning cannot be done on-line in many complex robotic problems, where quick responses are frequently required. In this paper, we propose an alternative approach based on case-based policy-learning which integrates deliberative reasoning through Automated Planning and reactive response time thought reactive planning policies. The method is based on learning planning knowledge from actual experiences to obtain a case-based policy. The contribution of this paper is two fold. First, it is shown that the learned case-based policy produces reasonable and timely responses in complex environments. Second, it is also shown how one case-based policy that solves a particular problem can be reused to solve a similar but more complex problem in a transfer learning scope.

Keywords: Automated Planning; Case-Base Reasoning; Robotics; Control Systems; Planning and Execution

1. Introduction

Many efforts have been conducted in robotics research for generating control systems that offer long-term reasoning capabilities. These capabilities allow autonomous agents to solve complex problems in dynamic or stochastic environments, which are similar to the real world. In these real world problems, the "world state" changes over time either by exogenous events or by the actions performed by the agent. Ad-

ditionally, these problems involve situations in which it is not possible to observe the entire state of the environment and where the execution of an action can generate unexpected states. For instance, a robot may not know the real traversability of a terrain until it is crossing it, or a logistic system may not be able to predict future traffic congestions nor even to be sure about current traffic conditions when defining a route for different trucks. Therefore, these problems require a real-time interaction, the constant monitoring of the world state in order to choose the appropriate action, and the recovery from unexpected situations.

The selection of decisions by an autonomous agent might be seen like a multi-criteria decision making problem [33]: the agent have to select the best decision to make according to different criteria. However, there are two main perspectives to generate a control system for autonomous agents in such stochastic and dynamic scenarios. At one extreme, deliberative reasoning behaves more like they are thinking, by searching through a space of behaviours, maintaining an internal state, and predicting the effects of the actions. Such deliberative reasoning is being increasingly used in robotic systems because it employs some form of forward projection (reasoning in depth about goals, preconditions, resources, and timing constraints) and provides reasonable responses in situations unforeseen by the designer. Automated Planning (AP) is one of the most extended techniques currently. AP studies the generation of action sequences – plans – for problem solving. A problem in AP is defined by a state-transition function describing the dynamics of the environment, the initial state and the goals to be reached. Although several methods based in AP has been successfully combined in robot control architectures to solve real world problems such as planning Mars exploration missions [4], managing fire extinctions [10] or controlling underwater vehicles [41], these solutions have been developed for specific problems including some knowledge that simplifies the deliberative reasoning.

Designing a deliberative system is not really complex but it needs high computational effort to generate solutions (i.e., plans). This computation effort increases with the complexity of the environment and the number of goals to be reached [11,23]. Additionally, it is important to note that, traditionally, planning techniques have been applied to "static" domains, i.e., domains in which the system has unlimited amount of time to solve each problem, and during this time, the "world state" does not change. However, in on-line planning, a plan is generated at the beginning of the execution, but also when there exists differences between the expected and the observed state of the world (a situation that we call unexpected situation) during the execution. In this case, the previous plan must be replaced with a new one [a]. Then, if there is a large number of unexpected situations during the execution, the use of such deliberative reasoning is unaffordable.

[a]There exists two strategies for obtaining this new plan: one is simply to replan from scratch and the other is to modify the existing plan to the new context [16]. In this paper, we focus on the first of these strategies.

On the other hand, reactive systems simply retrieve pre-defined behaviours similar to reflexes without maintaining any internal state. For instance, reactive reasoning based on Subsumption Architectures [8,9,6,5] is built using a control layer set, where layers are interconnected using signals. Each layer defines a primitive behaviour. During each execution step, one layer is chosen depending on the information perceived by the sensors of the agents. Reactive systems require much less computational effort that deliberative ones, but they are "mostly" blind with respect to the future; they usually ignore the impact of the selected behaviours on the next reasoning processes. Designing a reactive system can be a complex and time-consuming endeavour because of the need to pre-code all of the behaviours of the system for all foreseeable circumstances.

Regardless of the reasoning system used to build autonomous agents' behaviours, traditional planning systems operate under the assumption that planning for each new problem starts from scratch, thus disregarding any knowledge they may have gained while planning in previous problems. However, many robotic problems have similarities with each other. In such cases, some form of *knowledge transfer* or *transfer learning* between problems would be desirable. The core idea of transfer is that experience gained in learning to perform one task can support the learning of a related, but different, task. This is based on the idea of the world is *regular*, i.e., similar problems have similar solutions (so that if one starts from a solution to a similar problem, the solution will be found with little computational effort). Many examples in transfer learning can be found for classification, regression, and clustering [38], reinforcement learning [47] or planning [49,11,7].

But integrating learning with deliberative process in an autonomous system requires the use of software architectures able to control the different components, including behaviour execution, planning, monitoring, learning, etc. An example of such architecture is PELEA [39] which performs a cycle of constructing an initial plan to achieve certain goals, monitoring the plan execution, analyzing deviations from the original plan, re-planning when unexpected situations are found, and executing the new plans. This cycle requires the system to sense, interpret and deliberate about goals to be achieved, available actions, taking into consideration changes in the current world state, and resource or environmental constraints. The generation of the initial plan and the re-planning process are performed using AP, which is time consuming. Thus, one requirement of the architecture is the ability to learn planning knowledge from the experience to reduce the response time in future unexpected situations.

In this paper, we propose the learning of case-based policies from both the transfer of previously learned policies and the use of experiences produced by the execution of plans built through deliberative processes. To achieve this, we propose a novel Case-based planning [51,36,7] (CBP) algorithm to work in an on-line setting. CBP is planning as memorizing and involves reusing previous plans and adapting them to suit new situations. In this way, CBP can exploit regularities in the problems being solved, and thus potentially greatly increase the efficiency. The goal of

CBP is to improve the efficiency of planning systems by avoiding the repetition of the planning effort whenever it is not strictly necessary. However, under certain conditions, reusing plans has the same or even higher worst-case complexity than planning from scratch (e.g., when we deal with large case bases, or when we deal with the adaptation of large plans). Furthermore, in dynamic environments, with a high level of re-planning situations, the adapted plan is rarely fully executed: only the first actions before a new unexpected situation happens. Our approach is also based on reusing previous plans, but it does not store entire plans as other approaches do [13,44,48], but partial plans (i.e., the first actions of a plan). The number of actions of these partial plans is defined by a parameter which may be heuristically computed, as will be described later in this paper. Additionally, it does not accumulate one plan after another [30,13,48], but it includes a selection strategy for deciding which solutions to keep and which to discard based on the similarity between the new case and the nearest neighbor in the case base. The storing of partial plans and the proposed selection strategy reduce both the size of the case base, and the computational effort required to adapt the plans to the new situations.

In this paper, the proposed CBP algorithm has been integrated in the PELEA Architecture resulting in the CB-PELEA Architecture. In this way, the new architecture proposed combines the deliberative reasoning and a more reactive one (learned from the deliberative by CBR) by exploiting the benefits of both kinds of methods: reasonable responses in unexpected situations and timely responses. Additionally, we reuse the case-based policy from a past problem in order to solve a new one. In this way, in the new problem the number of re-planning steps (i.e., the number of times the high-level planner is invoked) is reduced from the beginning of the execution process (reducing, at the same time, the computational effort). Therefore, the contribution of this paper is twofold: (i) it is shown that the learned case-based policy produces reasonable and timely responses in complex environments, and (ii) it is also shown how one case-based policy that solves a particular problem can be reused to solve a similar problem.

It is important to point out that the case-based planner proposed in this paper is inside the category of planners which solve dynamic and stochastic problems by using deterministic planning and replanning. These systems are known as replanners. In general, deterministic planners discard any probabilistic information when proposing plans and, hence, they are more likely to produce dead ends. In our approach, if the deterministic planner used produces plans that lead to dead-end states, then the resulting case-based policy learned from these plans will lead to dead-end states too. This may be seen as a dramatic effect which precludes its application to real world problems. However, replanners systems based solely on deterministic planning have been successfully used in real-world domains known for unexpected outcomes [40,29,12,34]. In fact, deterministic planning is the most preferable option used in execution-monitoring-planning loops. Deterministic planners has advantages in computation time and execution efficiency, while probabilistic planners tend to produce overly cautious and thus overly long plans [12,27]. Addi-

tionally, in real-world problems the transition model (i.e., the probabilities of the actions' effects) are unknown and replanners can deal with these problems without a transition model and without a random exploration of the state space in search of better and better policies [3,1].

Therefore, within the category of replanners is where this paper proposes improvements. On one hand, instead of planning from scratch in every unexpected situation as classical replanners such as FF-replan do [52], we propose to learn a case-base that presents a faster response. On the other hand, there exist previous case-based planners in the literature; however, they differ from us because they store entire plans instead of partial plans [30,13,44], they do not include the monitoring of the executed plan [25,48,44], they simply accumulate in the case base all the new cases produced (i.e., without implementing any mechanism to decide whether it is advisable to store a new case or not) [30,13,48], or they focus on using the case base to guide the exploration process for a new solution plan rather than in adapting only the plan itself [51,11].

The paper is organized as follows. Section 2 introduces key definitions for AP, and CBR related to planning, and the languages used to encode domains and problems. Section 3 shows a detailed description of the proposed architecture. Section 4 shows the learning technique to learn case-based policies. Section 5 describes the general control flow of the architecture. Section 7 provides results in the deployment of the architecture in three domains related with robotic tasks: Rovers, Gold-Miner and Barman. In Section 8, we describe some works related with our approach. Finally, in Section 9 we conclude and introduce future work.

2. Background on Automated Planning and Case-Base Reasoning

In this section, some background regarding AP and CBR related to AP is presented. In particular, the Classical Automated Planning formalization of a planning task, the languages (PDDL and PPDDL) used to encode problems and domains, and the main CBP principles are briefly described.

2.1. *Planning Formalization*

In this paper, CB-PELEA receives as input a deterministic planning task and, hence, it discards any probabilistic information. This deterministic planning task can be defined as a tuple $\Phi = (T, O, F, A, I, G)$.

- T is a finite set of types.
- O is a finite set of objects. Each object $o \in O$ is associated with a type $t \in T$.
- F is a finite set of literals (also known as facts). A literal $f \in F$ is composed of a finite set of objects, $o \in O$. Using the objects in the planning task, O, planners instantiate all predicates obtaining the set of grounded literals F.
- A is a finite set of grounded actions derived from the action schemes

of the domain, where each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$. $Pre(a_i), Add(a_i), Del(a_i) \subseteq F$, $Pre(a_i)$ are the preconditions of the action, $Add(a_i)$ are its add effects, and $Del(a_i)$ are the delete effects. $Eff(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action. Besides, each action a_i has an associated non-negative integer cost, $cost(a)$ (the default cost is one).

- $I \subseteq F$ is the initial state.
- $G \subseteq F$ is a set of goals.

A state s is a subset of positive grounded literals, $s \subset F$, representing the literals which are currently true. Applying an action a in a state s_i can be defined as $s_{i+1} = (s_i \setminus Del(a)) \cup Add(a)$. An action a is applicable in s_i, if $Pre(a) \subseteq s_i$. A plan ϕ for a planning task Φ is a set of actions (in the common case a sequence) $\phi = (a_1, \ldots, a_n), \forall a_i \in A$, that transforms the initial state I into a state s_n where $G \subseteq s_n$. This plan ϕ can be executed if the preconditions of each action are satisfied in the state in which it is applied, i.e. $\forall a_i \in \phi\, Pre(a_i) \subseteq s_{i-1}$ such that state s_i results from executing the action a_i in the state s_{i-1}, considering s_0 as the initial state I. The cost of the solution is the sum of the action costs.

2.2. *Planning languages*

The Planning Domain Definition Language (PDDL) [17] is based on first-order logic where each atom of information is defined using a predicate, inspired by STRIPS formulations of Automated Planning problems. This language separates the description of actions which characterise domain behaviours from the description of specific objects, initial state and goals which characterise a problem to solve. According with this, a planning problem is defined by the pairing of a problem description with a domain description. An example of a deterministic planning action is shown in Figure 1 for the Rovers domain. It has three sections: parameters (three variables for the rover ?x, the origin waypoint ?y, and the destination waypoint ?z), the preconditions or the that must be true for the action can be executed (the rover ?x must be located in waypoint ?y, it is available, it can traverse from waypoint ?y to the destination waypoint ?z, and the waypoint ?z must be visible from waypoint ?y), and the effects (the rover is not in waypoint ?y, it has navigated to waypoint ?z).

However, we want to simulate stochastic effects in the environment. In order to do this, the Probabilistic Planning Domain Definition Language (PPDDL) [54] is used. PPDDL was the standard representation language that allowed to describe Automated Planning problems in stochastic domains in the first IPC with a non-deterministic track [55]. This language is an extension of PDDL that support actions with probabilistic effects and probabilistic literals on the initial state. Each set of literals is defined with a probability p. Equation 1 shows the structure of probabilistic effects on PPDDL, where $\forall e_i \; \exists \; p_i$.

```
(:action navigate
:parameters (?x - rover ?y - waypoint ?z - waypoint)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x)
                    (at ?x ?y)
                    (visible ?y ?z))
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Figure 1. PDDL representation for the action *navigate* from Rovers Domain.

```
(:action navigate
:parameters (?x - rover ?y - waypoint ?z - waypoint ?we1 - waypoint
             ?we2 - waypoint)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x)
                    (at ?x ?y)
                    (visible ?y ?z))
 :effect (probabilistic 0.75 (and (not (at ?x ?y)) (at ?x ?z))
                        0.10 (and (not (at ?x ?y)) (at ?x ?we1))
                        0.10 (and (not (at ?x ?y)) (at ?x ?we2))))
```

Figure 2. PPDDL representation for the action *navigate* from Rovers Domain.

$$(probabilistic\ p_1\ e_1\ p_2\ e_2\ \ldots\ p_{n-1}\ e_{n-1}\ p_n\ e_n) \tag{1}$$

Obviously, $p_i >= 0$ and $\sum_{i=0}^{n} p_i \leq 1$. If the sum of the probabilities is lower than 1, it is assumed that there is a probability $P = 1 - \sum_{i=0}^{n} p_i$ that the action does not generate any outcome. Figure 2 shows the traslated deterministic action in Figure 1 to the corresponding probabilistic action. In this action the robot *?x* will move to location *?z* with probability 0.75, it will move to location *?we1* with probability 0.1, and it will move to location *?we2* with probability 0.1. Finally, the action does not be executed with probability 0.05.

2.3. *Case-based Planning*

CBR is a field of AI that uses past experiences to solve new problems [2]. CBR have a case base B of previously solved problems constituting the experience of the system. Every case is a pair consisting of a problem description and some problem solving knowledge that could help to solve future similar problems. The problem description comprises any features that is relevant for defining a task in any representation paradigm. The problem solving knowledge can take the form of the solution to the

problem or it can contain some reasoning information, e.g., about how the solution has been found previously.

CBR has been applied widely to AP resulting in Case-based Planning (CBP) [7]. There are two categories depending on the information stored in each case and the use made of this information: transformational CBP and derivational CBP. In transformational CBP, each case contains a problem description (composed of a state $s \subset F$ and a set of goals $g \subset G$) and the solution plan for that problem (i.e., ϕ). These plans are reused in the new situations by making suitable changes when required. In contrast, cases in derivational CBP contain a *derivational trace* (i.e., a sequence of computational steps a planner followed to generate a plan) rather than a plan as in transformational CBP. This trace is used to provide a guidance in the new planning processes (it reduces the search space by pruning fruitless past choices). In this paper we focus on transformational CBP and the proposed approach can be considered inside this category. Section 4 describes the case's structure defined in this paper, and how the case-based policy is learned and used. But first, we describe the planning and execution architecture where the learning approach is integrated and that is used in the empirical evaluation.

3. Planning and Learning Architecture

This section describes the proposed planning architecture, CB-PELEA, based on PELEA [b]. It integrates planning, execution, monitoring, re-planning, and learning. As shown in Figure 3, CB-PELEA is composed of four sub-modules that exchange a set of items during the execution steps. The main items that we have used are: *state*, abstracted high-level state (i.e., current world information); tuples <*meta-state, actions*>, learning examples to be used by the learning component to acquire knowledge for future planning episodes; *domain*, definition of the model for high-level planning; *problem*, composed of the initial state and a set of goals to achieve; *plan*, a set of ordered actions resulting from the high-level planning process; *action*, a single piece of the plan. Next, each module of the architecture is described.

- **Monitoring**: it is the central module of the CB-PELEA architecture. It synchronizes communications between other modules (Execution, Decision Support and Learning). This module monitors action execution dispatching next action to Execution Module, requesting for a new plan to Decision Support Module and checking for differences between the expected state and the observed state of the environment sent by Monitoring Module. If an observed state is not valid, this module will have to start another planning episode to generate a new plan according to the observed state. Section 5 provides a detailed description of the Monitoring algorithm.

[b]The architecture described here is an instantiation of the original version [39]. In this paper, only some modules are used and a learning module is added.

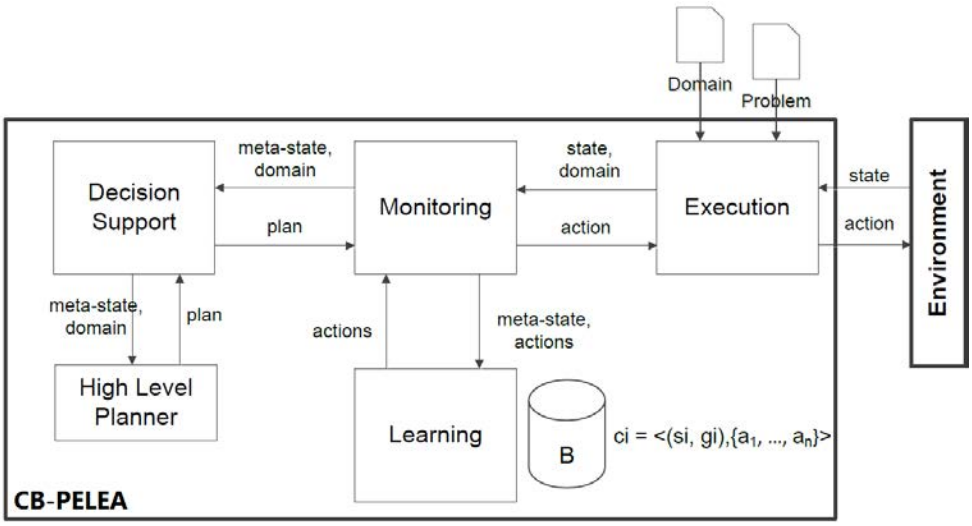

Figure 3. CB-PELEA architecture proposed to planning and execution for mobile robot control.

- **Execution**: this module executes actions in the environment. Besides, it is responsible for initiating the work of CB-PELEA by sending to Monitoring a particular problem and domain definition to be solved. Both the problem and the domain are described using the PDDL language described in Section 2. In this work the environment and its probabilistic effects are simulated by MDPSim and the PPDDL language as detailed in Section 7.
- **Decision Support**: this module generates a plan of actions by the invocation of a high-level planner (e.g., Fastdownward [26]). Additionally, when the Monitoring informs about a discrepancy between the observed state and the expected planning state, the Decision Support will also invoke the high-level planner generating a new plan for the new state. This module can be configured to use two different planners: the first one is used for planning from scratch when there is no plan; the second one can be configured for re-planning or repair depending of the planner used.
- **Learning**: this module infers knowledge from the experience gathered by the high-level planner during the plan execution. The knowledge can be used either to update the domain planning model, to improve the planning process (for instance, learning heuristics), or to reduce the action response time by the learning of a case-based policy. In this work, the learning module gathers $<$*meta-state,actions*$>$ pairs to build reactive action policies as will be described in Section 4.
- **High-level planner**: it is the planner system itself (e.g., Fastdownward [26]). It is invoked by the Decision Support module every time a plan is required.

4. Automated Case-Based Policy Learning Algorithm

Since on-line planning and planning under uncertainty are both unrealistic for large-scale [21], we propose to follow an approach based on CBR techniques to learn a case-based policy and, in this way, reduce the response time of CB-PELEA when it faces unexpected situations. According with the literature, solving a problem using CBR implies a set of steps: (i) defining or obtaining a case definition, (ii) defining a metric for measuring the similarity of the current problem to previous ones stored in the case base, (iii) deciding whether to retain new cases or not, (iv) retrieving one or more similar cases to the new problem, and (v) attempting to reuse the solution of the retrieved cases. Accordingly, this section describes each of these steps adapted to planning. In this way, Section 4.1 presents the case-base representation used in this paper. Section 4.2 describes the similarity metric used to measure the distance between the current state (in our case, a meta-state) and those stored in the case base. Sections 4.3 describes the retention phase adapted to planning. Finally, Section 4.4 describes the retrieval and reuse phases.

4.1. *Case-Base Representation*

The policy to be learned is a mapping from the tuple $< state, goals >$ to a sequence of actions, $\pi : (S, G) \rightarrow A^{\mu}$, where $A^{\mu} = \{a^{i}, \ldots, a^{\mu}\}$ is a sequence of μ actions to perform from state $s \subset F$ to satisfy in the future the goals $g \subset G$. It is important to note that A^{μ} is not the full plan ϕ (the entire sequence of actions) to perform from $s \subset F$ to satisfy the goals $g \subset G$. Instead, it could be considered as a *partial plan*: it is composed of the first μ actions of the full plan ϕ. In this way, the policy $\pi : (S, G) \rightarrow A^{\mu}$ is represented as a Case Base, $B = \{c_1, c_2, \ldots, c_{\eta}\}$ where every case c_i consists of a pair state-goals $m_i = (s_i, g_i)$ and with the associated sequence of actions $\{a_i^1, \ldots, a_i^{\mu}\}$. Thus, $c_i = \{m_i, \{a_i^1, \ldots, a_i^{\mu}\}\}$, where the first element represents the case's problem part and corresponds to the meta-state $m_i = (s_i, g_i)$, and the final element, $\{a_i^1, \ldots, a_i^{\mu}\}$, depicts the case solution (i.e., the sequence of μ actions expected when the agent is in the state s_i and wants to reach the goals g_i).

Storing full plans has two main drawbacks: the storing of long plans or plan created in complex dynamic environments can lead to increase memory consumption and response times [43]. Furthermore, in dynamic environments with a high level of unexpected situations, the plan stored is rarely fully executed, only the first actions until a new unexpected situation happens. On the opposite, storing only one action for each $< state, goal >$ pair has in turn two main drawbacks: it increases the number of requests to the case base since, for each request, only one action to be executed is recovered, and it is partly lost the forward reasoning provided by the planner (the planner provides a sequence of actions to be executed for a state $s \subset F$ in order to satisfy the goals $g \subset G$ and not only one action).

Therefore, the proposed case's structure has three main advantages: (i) it reduces the memory consumption, (ii) it reduces the number of requests to the case base

since, for each request, μ actions are recovered to be executed instead of only one, and (iii) it remains part of the forward projection provided by the planner. In this way, the policy, π, can be seen as an element that compiles a vast planning experience, in such a way that the agents can use it to quickly make decisions.

4.2. *A Distance Metric for Automated Planning*

We follow an adapted version of the distance proposed by García et al. [22], which is at the same time a simplification of the Relational IBL distance metric [32]. To compute the distance between two meta-states, m_1 and m_2, let us assume that there are n predicates in m_1, $m_1 = \{p_1^1, , \ldots, p_n^1\}$, and m predicates in m_2, $m_2 = \{p_1^2, \ldots, p_m^2\}$. For each predicate $p \in m_1$ and $p \notin m_2$ and viceversa, the distance between the two meta-states, $d(m_1, m_2)$ is increased by the weight factor associated with that predicate. For each predicate, $p \in m_1$ and $p \in m_2$, we compute the distance between the set of literals, $P(m_1)$, of predicate $p \in m_1$, and the set of literals, $P(m_2)$, of predicate $p \in m_2$ as defined in Equation 2.

$$d(m_1, m_2) \mathrel{+}= w_p \sum_{k=1}^{K} \min_{p \in P(m_2)} d(p_k^1, p) \tag{2}$$

where K is $|P(m_1)|$, p_k^1 returns the kth literal from the set $P(m_1)$, and w_p is the weight factor associated to predicate p. Basically, this equation computes for each literal p_k^1, the minimal distance to every literals in $P(m_2)$. Finally, the distance between two literals of the same predicate is computed comparing the arguments as defined in Equation 3.

$$d(p_l^1, p_h^2) = \frac{1}{J} \sum_{j=1}^{J} \delta(\arg_j p_l^1, \arg_j p_h^2) \tag{3}$$

where J is the number of arguments, $\arg_j p_l^1$ returns the jth argument of literal p_l^1, $\arg_j p_h^2$ returns the jth argument of literal p_h^2, and δ returns 0 if both values are the same, and 1 if they are different. To save time computing the distance between invariant parts of the meta-state, we do not have to take into account the static predicates, i.e., the predicates which are never in the effects of the domain actions.

Figure 4 illustrate the computation of the proposed distance. It shows two meta-states, $m_1 = \{p_1, \ldots, p_5\}$ and $m_2 = \{p_1, \ldots, p_4\}$. We assume all the predicates have a weigth factor set to 1. First, we increase the distance for each predicate $p \in m_1$ and $p \notin m_2$ and viceversa (italic predicates in Figure 4). In this case, we increase the distance to 5.

Later, we compute the distance for each predicate, $p \in m_1$ and $p \in m_2$ (bold predicates in Figure 4). That is, we compute the distance between predicate $p_1 \in m_1$ and $p_1 \in m_2$ (resulting 0 since they are the same), and the distance between $p_2 \in m_1$ and $p_3 \in m_2$ (resulting 0.5 since they differ in one argument). Therefore,

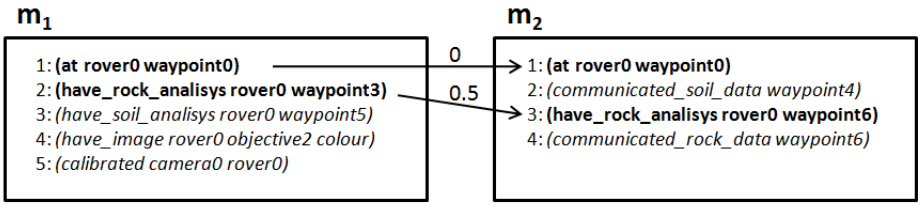

Figure 4. Graphic illustration of how the distance is computed between the meta-states m_1 and m_2.

the final distance between m_1 and m_2 is $d(m_1, m_2) = 5.5$. Therefore, in contrast with García et al. [22], we do not compute this distance taking into account only the same predicates in both meta-states. If we proceed in such way, the final distance between the meta-states in Figure 4 would be $d(m_1, m_2) = 0.5$, which does not really reflect how different the meta-states are.

However, given this similarity metric, the distance between two literals depends on the similarity between the names of the sets of objects, i.e., the distance between two meta-states that are exactly the same but with different object names is greater than 0. To partially avoid this problem, the object names of every meta-state are renamed. Each object is renamed by its type name and an appearance index. The first renamed objects are the one that appear in literals of the state, followed by the objects that appear in the goals. Thus, we try to keep some kind of relevance level of the objects to find a better similarity between two literals. For example, for the meta-state: *(can_traverse waypoint2 waypoint5), (can_traverse waypoint5 waypoint4), (at rover3 waypoint2), (goal_at rover3 waypoint4)* we obtain the corresponded adapted meta-state: *(can_traverse waypoint0 waypoint1), (can_traverse waypoint1 waypoint2), (at rover0 waypoint0), (goal_at rover0 waypoint2)*. Note that, in this way, the meta-state *(can_traverse waypoint5 waypoint3), (can_traverse waypoint3 waypoint2), (at rover8 waypoint5), (goal_at rover8 waypoint2)* has the same adapted meta-state.

4.3. *Retention: Generation of Planning Cases*

Our approach requires a set of cases in its case-base in order to work. In this paper, we propose to learn such cases in an on-line manner using a different perspective with respect to the traditional ones [19].

We illustrate our approach through the parent-child analogy, where a teacher takes the role of the parent and the learner agent takes the role of the child. At the beginning, the child does not know much about the world, hence, he/she needs to explore. While learning, most of the situations are new for him/her. In these unknown situations, he/she prefers to take actions advised by the parent. The child incorporates the parent's knowledge into his/her own knowledge about the world. Translating this analogy into the CB-PELEA architecture (Figure 3), the parent

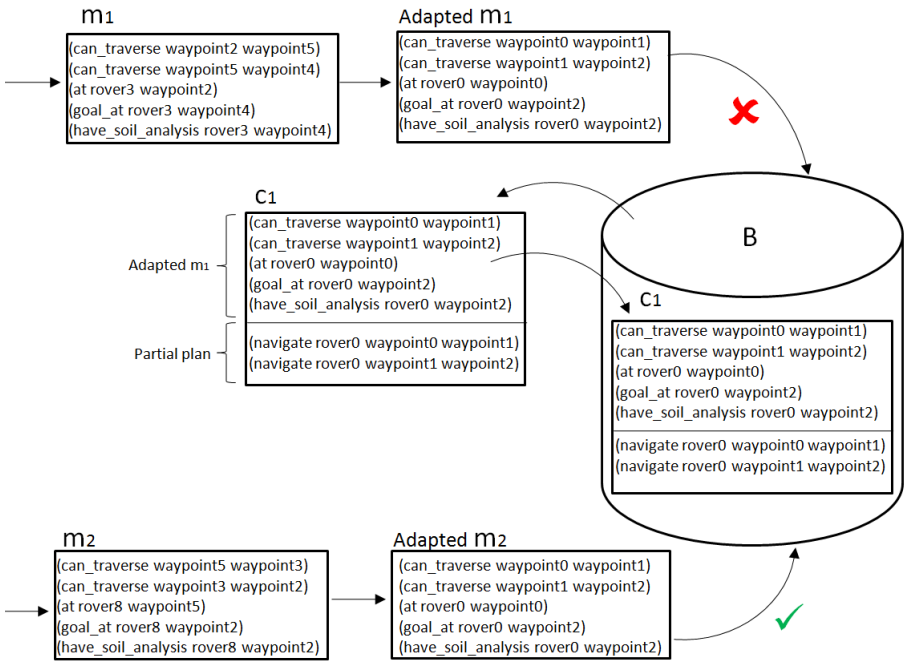

Figure 5. Graphical representation of plan acquisition.

(teacher) corresponds to the Decision Support module which invokes the corresponding high-level planner, and the Learning module corresponds to the child. Therefore, the performance of the child is heavily limited by teacher ability. For instance, if the high-level planner generates plans which lead to dead ends, then the learned case-based policy will generate dead ends too.

Traditionally, case-based approaches use a *density threshold*, θ, in order to determine when a new case is unknown. When the Learning module (the child) receives a new meta-state, m_q, and its distance to the nearest neighbour in B is greater than θ, the case is considered unknown, and no case is retrieved [20]. Consequently, the sequence of actions to be performed from that meta-state is unknown. Then, a new plan is required to the high-level planner (the teacher). After that a new case is composed using the perceived meta-state, m_q, and the first μ action of the new plan generated. This new case is added to the case base B. In such a way, starting with an empty case-base, the learning algorithm continuously increases its competence by storing new experiences.

The objects of the new cases added to the case base are renamed using the same criteria that the described in Section 4.2. The purpose of this is to avoid the addition of cases that are very similar but with different object names. This is illustrated in Figure 5.

Let us assume that initially the case base in Figure 5 is empty, $\theta = 1$, and

$\mu = 2$. Let us also assume that we have first received a meta-state m_1. Then, it is adapted, i.e., its objects are renamed. After that, the distance between the adapted meta-state m_1 and its nearest neighbour in B is computed. Since the case base is empty, we consider this distance is greater than $\theta = 1$. Therefore, no partial plan is retrieved from B. The high-level planner is required to generate a plan for the adapted meta-state m_1. Then, a new case c_1 composed of the adapted meta-state m_1 and the first $\mu = 2$ actions of the new plan is added to the case base. Then, continuing with our example, let us assume that a new meta-state m_2 is received and adapted. The distance between the adapted m_2 and its nearest neighbour in B is 0, and it is not necessary the construction of a new case c_2 for its addition to the case base B.

It is important to note that by following this sequence of steps but without the adaptation processes, both meta-states, m_1 and m_2, would require the construction of its own cases, c_1 and c_2, and these cases would be added to the case base. However, both meta-states are exactly the same but with different object names.

4.4. *Retrieval and Reusing of a Plan*

When the Monitoring module from CB-PELEA receives a new meta-state, m_q, from the environment, it requests the learning module for a sequence of actions. The learning module retrieves the nearest neighbour of m_q in B, according to the distance metric defined in Section 4.2. If the distance is lower than the *density threshold*, θ, it returns the associated sequence of actions. It is important to be aware of the fact that this procedure prevents to store all plans in the case base, which is already impossible and searching for the applicable one is another impossible task.

Figure 6 shows an example of retrieval and reusing of a plan stored in the case base.

First, the meta-state, m_1, is received. Then, its objects are renamed. After that, the distance between the adapted meta-state m_1 and its nearest neighbor in B is computed. This distance is lower than θ, and then the partial plan corresponding to the nearest neighbor case is returned, i.e., *(navigation rover0 waypoint0 waypoint1), (navigation rover0 waypoint1 waypoint2)*. However, it is necessary to adapt the object names in the retrieval plan to the specific names of the received meta-state, m_1. Fortunately this is straightforward since in the adaptation process of m_1, we replace *rover3* with *rover0*, *waypoint2* with *waypoint0*, *waypoint5* with *waypoint1*, and *waypoint4* with *waypoint2*. Now just have to undo these replacements. Finally, the returned adapted partial plan is: *(navigation rover3 waypoint2 waypoint5), (navigation rover3 waypoint5 waypoint4)*.

5. Detailed Description of the Control Flow

The algorithm used to control the flow of execution is depicted in Figure 1.

It controls the execution of the different modules of the CB-PELEA architecture. The algorithm receives as inputs a planning task $\Phi = (T, O, F, A, I, G)$, the *density*

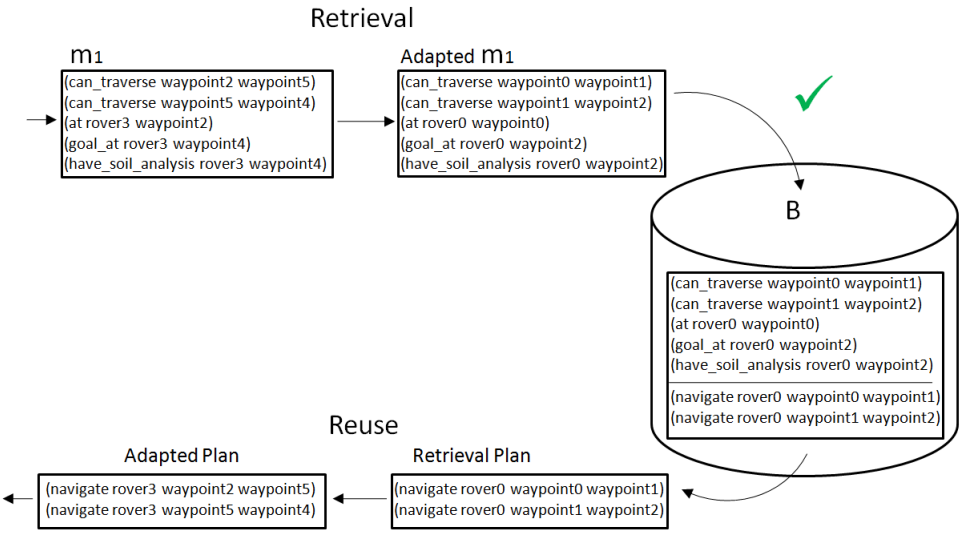

Figure 6. Graphical representation of plan retrieval and reusing when a meta-state m_1 is received.

threshold θ, the length of the partial plans stored in the case base, μ, and a maximum number of episodes to execute each, H. First, the algorithm initializes the variables used: the variable *episodes* used to control the number of episodes executed, the case base B, the variable s used to control the state of the world, the partial plan $partialPlan$, and the variable $finish$ used to control the termination of a running episode (line 2). Then, the loop begins. This loop controls wheter the goals are reached, i.e., if the goals are in the current state, s, and if it has been reached a finish condition, i.e., a plan is not found given the current state of the world (line 4). If the partial plan is empty or the state s sensed from the environment is an unexpected state [c], the algorithm requests a partial plan to B (line 6), i.e., it requests the partial plan of the case c_i for which $d(m_q, m_i) = \min_{1 \leq j \leq \eta} d(m_q, m_j) \leq \theta$, where η is the size of the case base.

However, if the distance of the current meta-state to any meta-state in B is larger than θ, no case is retrieved (line 7). In this case, the algorithm replans, i.e., it obtains a new plan from s to reach the goals G (line 8). Then, if the plan is not empty, a new case, c^{new}, is added to the case Base B (line 10-11). Finally, the next action a of the partial plan is executed into the environment (lines 13-14) and, after its execution, the new state of the world is sensed (line 15). It is important to note that the greater μ, the lower is the number of requests to the case base B (as long as the transited states are valid states). However, although the number

[c]In our case, if the observed state of the world s differs from the expected state of the world given the execution of an action a, then we consider that s is an unexpected state. For instance, if the robot has performed the action of navigate from *waypoint0* to *waypoint1* but it has reached the *waypoint2* instead of the expected *waypoint1*, then we consider it has reached an unexpected state.

Algorithm 1: Automated Case-Based Policy Learning Algorithm.

Data: $\Phi = (T, O, F, A, I, G)$, θ, μ, H
Result: The policy π_B^θ.

```
1  begin
2      Set episode = 0, B = ∅, s = I, partialPlan = ∅, finish = false;
3      repeat
4          while G ⊄ s AND !finish do
5              if partialPlan = ∅ OR unexpectedState(s) then
                   /* Retrieve and Reuse Plan (Section 4.4)          */
6                  partialPlan = retrieveReusePlan(B, m_q = (s, G));
7                  if partialPlan = ∅ then
                       /* No plan is retrieved from the case base.
                          Invocation to the highLevelPlanner and
                          acquisition of a new case (Section 4.3)    */
8                      partialPlan = highLevelPlanner(T, O, F, A, s, G, μ);
9                      if partialPlan ≠ ∅ then
10                         Create a new case, c^new =< (s, G), partialPlan >;
11                         retainCase(B, c^new, θ);
12             if partialPlan ≠ ∅ then
13                 a = getNextAction(partialPlan);
14                 executeAction(a);
15                 s = getSensors();
16             else
17                 finish = true;
18         finish = false;
19         episode = episode + 1;
20     until episode = H;
21     return π_B^θ;
```

of requests is reduced, the memory consumption is increased. Therefore, a correct selection of this parameter is required. In Section 7, a solid perspective is given on the automatic definition of the parameters θ and μ.

6. Cased-Based Reuse Across Similar Problems

The idea proposed here is similar in spirit to the proposed by Fernández et al. [14] in Reinforcement Learning. In our case we adapt this idea to work in planning tasks. We assume a past Cased-Based policy $\pi_{B_{past}}$ associated to the Case-Base B_{past} that solves a problem $\Omega_{past} =< O_{past}, F_{past} >$ composed by a set of objects O_{past} and a set of literals F_{past}. We w-ant to learn a new Case-Based policy $\pi_{B_{new}}$ associated to

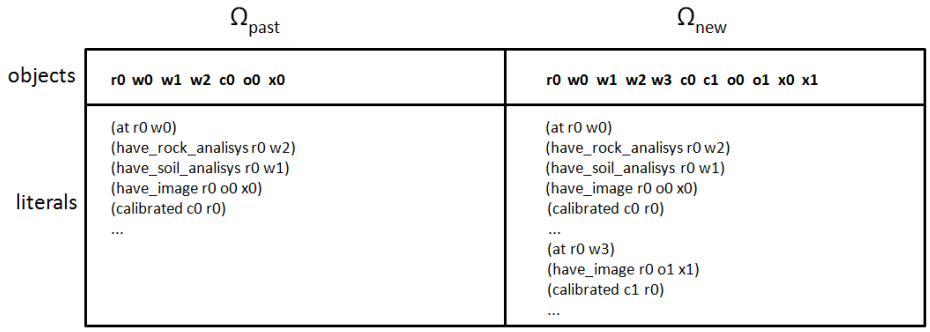

Figure 7. Objects and literals for the problems Ω_{past} and Ω_{new}.

the Case-Base B_{new} that solves a new problem $\Omega_{new} =< O_{new}, F_{new} >$ composed by a set of objects O_{new} and a set of literals F_{new}, where $O_{past} \subseteq O_{new}$ and $F_{past} \subseteq F_{new}$. We also assume the actions that can be performed are the same in both problems. To reuse the past policy $\pi_{B_{past}}$ in the new problem Ω_{new} we need to find a function $\rho : P(m_{new}) \rightarrow P(m_{past})$ that maps the literals of a meta-state m_{past} in problem Ω_{past} to a meta-state m_{new} in problem Ω_{new}, where $P(m_{past}) \subseteq F_{past}$ is the set of literals of meta-state m_{past} and $P(m_{new}) \subseteq F_{new}$ is the set of literals of meta-state m_{new}.

In planning, this mapping is derived from the semantic of the literals. To illustrate this, we use Figure 7. It shows the objects and the set of literals of two problems in the Rovers domain, Ω_{past} and Ω_{new}. Each literal in Ω_{new} maps to the literal of Ω_{past} in the same row. For instance, literal *(at r0 w0)* in Ω_{new} maps to the literal *(at r0 w0)* in Ω_{past}. The literals in Ω_{new} that does not have equivalent in Ω_{past} are eliminated, as it is the case of *(calibrated c1 r0)*. The reason is that it involves information of new objects present in Ω_{new} but not in Ω_{past}.

If transfer learning is used, when the agent receives a new meta-state m_q it performs the partial plan of the case c_i for which $d(\rho(m_q), m_i) = \min_{1 \leq j \leq \eta_{past}} d(m_q, s_j) \leq \theta$, where η_{past} is the size of the case base B_{past}. If no case is retrieved, it performs the partial plan of the case c_i for which $d(m_q, m_i) = \min_{1 \leq j \leq \eta_{new}} d(m_q, s_j) \leq \theta$, where η_{new} is the size of the case base B_{new}. However, if the meta-state distance to any meta-state in B_{past} and B_{new} is larger than θ, a new case is added to the case base B_{new}.

7. Experimental Results

The following sections provide an evaluation of the proposed architecture over three benchmark domains from the International Planning Competition (IPC) set: Rovers from the IPC-3, Gold-Miner from the learning track of the IPC-6 and Barman from the IPC-7. All these domains are related to robotic tasks.

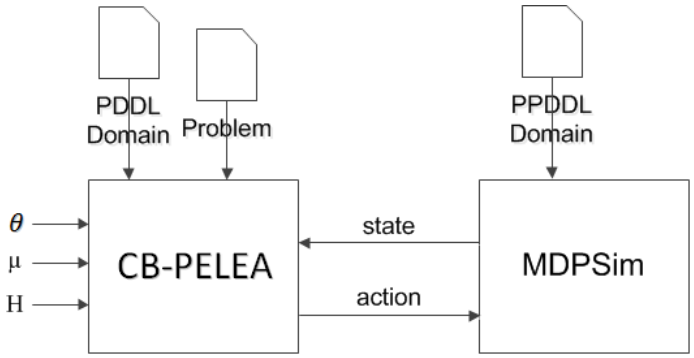

Figure 8. Diagram of the experiments configuration

7.1. *Experimental Scope*

The experiments have been conducted according with the configuration shown in Figure 8. Figure 8 shows CB-PELEA and the environment simulated by MDP-Sim. CB-PELEA receives as inputs the planning task Φ by the PDDL definitions of the domain and problem, and the parameters θ, μ and H as shown in Section 5. Besides, MDPSim receives as input the PPDDL definition of the domain which lets us introduce execution probabilities to the actions to simulate stochastic environments as described in Section 2 [d]. CB-PELEA sends actions to MDPSim, and MDPSim returns the resulting states from applying the probabilistic actions.

For each domain, we provide two sets of experiments with different goals. The first set is focused on demonstrating the advantages of using a cased-based policy and the second one is focused on demonstrating the advantages of reusing a case-based policy across different tasks.

In the first set, we present the results in three different problems of increasing complexity (i.e., increasing number of objects, literals, and goals). For each problem, we present the results collected from the use of the state-of-the-art case-based planner OAKPlan [44], from the planner FD-Replan based on FF-Replan [52], and from the use of PELEA in which the learning capabilities are enabled, i.e., the version of PELEA presented in this paper (we call this version of PELEA as CB-PELEA). It is important to point out that we are not trying to prove whether our algorithm is better than OAKPlan (since it stores different information and the retrieval, reuse and retention phases are also completely different), but instead we will take the results achieved with it as reference to analyze the performance achieved with our algorithm. Similarly to our approach, each case in OAKPlan consists of a description of the state and the goals to be reached, but it stores entire plans instead of partial plans. Additionally, each case in OAKPlan also stores the graph

[d]Such probabilities are unknown for CB-PELEA, which receives only a deterministic version of such domain

representation of the problem used to achieve an efficient retrieval, and the degree sequences of this graph. In the reuse phase, OAKPlan uses LPG-Adapt [16] to adapt the retrieval plan so that it solves the current problem. In our case, we propose a much simpler adaptation process in which the object names are renamed according to the current problem. On the other hand, FD-Replan is a simply approach which calls FastDownward on the planning problem and selects actions according to the plan until observing an unexpected state, upon which it replans by invoking to FastDownward again. Therefore, the only difference between FD-Replan and FF-Replan [52] is that FD-Replan uses FastDownward instead of FF. The results collected from FD-Replan are used to verify the advantages of using a case-based policy to solve the unexpected situations instead of creating a plan from scratch by invoking a high-level planner.

In the case of CB-PELEA, we test two different values of θ: the first one is $\theta = 0$ (i.e., a perfect match is required to reuse a case, otherwise a new case is constructed and added to the case base), and the second one is a value greater than 0 (i.e., it enables to reuse partial plans previously computed in cases that are similar, but not exactly equal, to the current state). This second value is computed heuristically as the mean distance between states during an execution of FD-Replan. A similar heuristic has been used sucessfully in previous CBR-Works [20]. It is important to note that when $\theta = 0$ the retrieved plans are supposed to be correct, but when $\theta > 0$ the retrieved plans may be incorrect. For this reason, the purpose is to analyze the influence of the parameter θ on the number of actions needed to reach the goals, and on the size of the case base. Additionally, for each value of θ, we test three different values of μ: $\mu = 1$ (i.e., each case stores only one action), a value of μ equal to the number of actions needed to reach the goals given the current state (i.e., each case stores the full plan), and a value selected heuristically. In this case, the proposed heuristic is computed as the mean length of the full plan (i.e., mean number of actions of the plan) divided by the mean number of replanning situations in the execution of the plan. That means, the mean number of actions that CB-PELEA is able to execute consecutively without replanning. A lower value of μ means to increase unnecessarily the number of requests to the case base, and a higher value implies an unnecessary memory consumption in storing longer partial plans that will not be executed completely (it is highly probable that a replanning situation appears before that happens). Therefore, the selection of these three values for the parameter μ has the purpose of analyzing the advantages of storing partial plans instead of only one action or full plans. Finally, the parameter H is set to $H = 50$ for the Rovers domain, it is set to $H = 20$ for the Goldminer domain, and $H = 50$ for the Barman domain. To get a fair comparison of performances in this first set of experiments, we have adapted OAKPlan to work in an on-line setting: for each run, both OAKPlan and CB-PELEA starts with an empty case base, and new cases are added as the learning process proceeds.

The results for different problem sizes and approaches are analyzed across seven dimensions: the accumulated cost measured as the number of actions in the executed

plan (we label as *Cost* the column in Table 1 and Table 2, which shows the results for this dimension), the accumulated time (seconds) used to solve the problem (*Time*), the average response time (seconds) per action (*Response*), i.e., the mean time between when CB-PELEA receives the current state and when sends the action to perform in that state, the number of times it is necessary to invoke the high-level planner (*Planner*), the number of actions of the retrieved and adapted plan that are not performed (*Discards*), the number of cases in the case base at the end of the learning process (*Cases*), and the final size (in megabytes) of the case base (*Size*).

In the second set of experiments, we show how the case-based policy learned in one problem can be reused to solve a similar problem. In this case, the purpose is to demonstrate how the response times in one problem can be reduced from the beginning of the learning process by the use of a case base learned previously in a similar problem. Therefore, in contrast with the first set of experiments, the execution of CB-PELEA starts with an empty case base B_{new}, but it uses the cases in B_{past} from solving a similar problem when deemed necessary.

Experiments were conducted on an Intel Xeon 2.93 GHZ Quad Core processor (64 bits) running under Linux. The maximum available memory for the planners was set to 2 GB, the maximum planning time for a problem has been set to 100 seconds and the maximum execution time has been set to 2500 seconds. The high-level planner used is Fast Downward [26]. This planner is also used both by OAKPlan and CB-PELEA when no case is retrieved from the case base. The tables and graphs show the means computed from five different executions.

7.2. *Rovers Domain Results*

Rovers Domain was designed for the sequential track of IPC-3 (2002) and it was inspired on the Mars exploration rovers missions where an area of the planet is represented as a grid of cells, called waypoints. The problems in this domain contain samples of rock or soil that can be collected by the robots. Each robot can traverse across different waypoints and can perform a set of different actions (analyze rock or soil samples, or take pictures of a specific waypoint). All data collected by the robots has to be sent to the lander, that is placed in a specific waypoint. Rock and soil samples does not disappear from the waypoint when a rover takes it. For this domain, we have designed an error model based on four failures which have been encoded into PPDDL: (i) there is a general error with probability 0.5 which prevent the execution of any action (i.e., the action is not executed), (ii) the camera can turn out of calibration with a probability of 0.1, (iii) there is a communication error in which the sample or image are lost with a probability of 0.1, and (iv) a navigation error happens when a rover moves to a different waypoint when is navigating with a probability of 0.2.

Table 1 shows the results for three different problems increasing in complexity using OAKPlan [44], FD-Replan, and CB-PELEA using different values for the parameters θ and μ. The results in Table 1 are presented across the seven dimensions previ-

ously described in Section 7.1: *Cost, Time, Response, Planner, Discards, Cases and Size*. The values for the parameters θ and μ are computed heuristically as described in Section 7.1. In this case, $\theta = 0.75$ and $\mu = 4$ for the simplest problem, $\mu = 5$ for the medium complexity problem, and $\mu = 6$ for the most complex problem. The problem size (*Problem Size*) is expressed as $\#waypoints \times \#objectives \times \#goals$.

Problem	Approach		Cost	Time	Response	Planner	Discards	Cases	Size
30 × 20 × 7	OAKPlan		88(4.8)	104(13.5)	1.2(0.4)	5(2.1)	31.4(3.5)	1399.2(55.7)	63.5(1.7)
	FD-Replan		**85**(13.6)	136(9.2)	1.8(0.2)	22(5.6)	-	-	-
	CB-PELEA	θ=0 μ=1	89(12.3)	123.1(7.2)	1.4(0.08)	4(1.8)	**0**	1573(56.9)	1.8(0.3)
		θ=0 μ=4	86(11.2)	77.4(4.2)	0.9(0.03)	5(0.8)	0.8(0.2)	1462(42.1)	3.1(0.4)
		θ=0 μ=full	87(10.1)	82.2(6.7)	1.1(0.06)	3(1.1)	32.4(10.5)	1603(76.9)	22.9(1.2)
		θ=.75 μ=1	98(9.1)	73.2(8.5)	0.9(0.02)	6(0.7)	**0**	1102(24.5)	**1.2**(0.2)
		θ=.75 μ=4	91(10.8)	**51.2**(2.4)	**0.4**(0.05)	5(0.9)	1.1(0.7)	**994**(53.7)	2.1(0.7)
		θ=.75 μ=full	94(8.4)	60.7(5.8)	0.6(0.08)	3(1.2)	39.3(11.8)	**982**(76.2)	14.3(1.3)
50 × 30 × 11	OAKPlan		98.2(5.4)	204.7(14.1)	2.1(0.6)	6(2.3)	38.3(4.2)	1988.4(67.2)	124.5(2.5)
	FD-Replan		97(14.1)	242.5(19.4)	2.5(0.2)	21.6(5.5)	-	-	-
	CB-PELEA	θ=0 μ=1	102.2(15.6)	234.6(18.4)	2.3(0.4)	1.8(1.5)	**0**	2104.2(115.1)	3.4(0.8)
		θ=0 μ=5	**96.2**(10.2)	168.2(21.3)	1.6(0.2)	2.7(1.2)	1.3(0.7)	2008.4(138.4)	5.1(1.2)
		θ=0 μ=full	104.2(13.4)	197.3(23.8)	1.9(0.3)	1.5(1.1)	37(9.7)	2194.5(114.8)	40.1(2.7)
		θ=.75 μ=1	110.4(17.5)	187.3(17.3)	1.8(0.4)	3.3(2.1)	**0**	1394.4(94.3)	**2.2**(0.8)
		θ=.75 μ=5	109.2(14.4)	**130.8**(22.5)	**1.1**(0.1)	2.8(0.7)	1.8(0.6)	**1272.4**(101.4)	3.2(1.1)
		θ=.75 μ=full	112.4(13.2)	156.8(14.7)	1.4(0.2)	2.3(0.9)	44(12.6)	1412.5(74.6)	25.8(2.1)
70 × 40 × 16	OAKPlan		160.3(9.7)	637(14.8)	3.9(0.5)	6(3.2)	61.4(2.5)	3244.7(78.7)	214.4(4.7)
	FD-Replan		**155.1**(20.4)	713.5(31)	4.6(0.2)	30(3)	-	-	-
	CB-PELEA	θ=0 μ=1	158.4(14.3)	663.9(18.3)	4.2(0.1)	4.1(3.1)	**0**	3247(161.2)	7.2(1.4)
		θ=0 μ=6	156.4(18.3)	599.4(19.7)	3.7(0.2)	6.1(2.1)	1.2(0.6)	3461(149.3)	11.4(2.2)
		θ=0 μ=full	159.2(17.2)	620.3(20.1)	3.9(0.2)	5.3(2.8)	58(14.6)	3597(201.1)	96.7(6.4)
		θ=.75 μ=1	170.1(21.1)	617.3(18.2)	3.7(0.2)	4.5(3.1)	**0**	**2049**(114.9)	**4.5**(0.8)
		θ=.75 μ=6	167.4(22.4)	**533.9**(14.7)	**2.8**(0.1)	5.2(2.2)	1.3(0.8)	2347(108.4)	7.7(1.6)
		θ=.75 μ=full	171.6(23.5)	595.9(17.8)	3.4(0.1)	4.3(1.6)	64(19.3)	2237(92.3)	60.1(5.4)

Table 1. Results in Rovers domain for different problem sizes and approaches across seven dimensions. Results for OAKPlan and CB-PELEA are measured after 50 episodes. Standard deviations are in brackets.

Table 1 shows three main results. First, if θ is equal to 0, the cost of solving the problems is similar both in FD-Replan and CB-PELEA; but if $\theta = 0.75$, this cost increases. For instance, for the problem $70 \times 40 \times 16$, the cost of FD-Replan is 155.1, but the cost when $\theta = 0.75$ is 170.1 for $\mu = 1$, 167.4 for $\mu = 6$ and 171.6 for μ equal to the entire plan. This is because if θ is equal to 0, a higher number of cases are added to the case base (a case for each new situation), and the retrieved plan fits exactly to the current situation. In this way, the cost is not affected when $\theta = 0$. However, if θ is greater than 0, the case base has a smaller number of cases. This implies that the same case (and, hence, the same partial plan) could be retrieved for different situations. This causes the retrieval of partial plans that are similar (but not exactly the same) to the required for the current situation and, hence, a higher

number of actions are required to solve the problem. In any case, although there is an increment in the cost when $\theta = 0.75$, the problems are successfully solved. Additionally, the cost of solving the problem by OAKPlan is also lower than the cost obtained by CB-PELEA. For instance, for the problem $70 \times 40 \times 16$, the cost obtained by OAKPlan is 160.3.

The second aspect for consideration is that CB-PELEA configurations with $\theta = 0.75$ take less time to solve the problem than those with $\theta = 0$. This is because when the case base has a smaller number of cases, the plans are retrieved faster. For instance, for the problem $70 \times 40 \times 16$, the time for $\theta = 0$ and $\mu = 1$ is 663.9 and it is reduced to 617.3 seconds when $\theta = 0.75$, the time for $\theta = 0$ and $\mu = 6$ is reduced from 599.4 to 533.9 seconds when $\theta = 0.75$ and, finally, the time for $\theta = 0$ and μ equal to the entire plan is reduced from 620.3 to 595.9 seconds when $\theta = 0.75$. OAKPlan requires a longer time to solve the problems than most of the configurations of CB-PELEA. This is because OAKPlan presents more complex retention and retrieval processes in which, additionally, it has to adapt entire plans.

Among the configurations where $\theta = 0.75$, the best times are those where μ is computed heuristically. On one hand, when μ is equal to 1 (i.e., each case stores only one action) it is necessary a request to the case base for each action to be executed. This increases the number of requests to the case base and, hence, this also increases both the total time and the response time per action required to solve the problem. For instance, for the problem $70 \times 40 \times 16$, the total time for $\theta = 0.75$ and $\mu = 1$ is reduced from 617.3 to 533.9 seconds, and the response time is reduced from 3.7 to 2.8 seconds when $\theta = 0.75$ and $\mu = 6$. On the other hand, if we store full plans in each case, the number of requests to the case base is reduced, but the time required to solve the problems is also higher than when μ is computed heuristically. This is because we have to adapt a higher number of actions than when we store partial plans. In this way, for the problem $70 \times 40 \times 16$, the total time for $\theta = 0.75$ and μ equal to the full plan is reduced from 595.9 to 533.9 seconds, and the response time is reduced from 3.4 to 2.8 seconds when $\theta = 0.75$ and $\mu = 6$. Additionally, when we retrieve and adapt full plans, many of the adapted actions are discarded because an unexpected situation happens before the plan is entirely executed. For instance, for the problem $70 \times 40 \times 16$, when $\theta = 0.75$ and μ equal to the full plan, it is adapted 64 actions of the retrieved plan that will be discarded. Instead, when $\theta = 0.75$ and $\mu = 6$, the number of discarded actions per retrieval is reduced to 1.3. Furthermore, storing full plans leads to an unnecesary memory consumption. For the problem $70 \times 40 \times 16$, when $\theta = 0.75$ and μ equal to the full plan, the size of the case base is 60.1 megabytes. Instead, when $\theta = 0.75$ and $\mu = 6$ the size of the case base drops to 7.7 megabytes. Instead, OAKPlan requires the largest case base, since it stores more information in each case. For the problem $70 \times 40 \times 16$, OAKPlan requires a case base of 214.4 megabytes.

The third aspect for consideration in Table 1 is that the use of CB-PELEA clearly outperforms the response time of FD-Replan and OAKPlan. In the case of FD-Replan, for the problem $70 \times 40 \times 16$, the total time is reduced from 713.5 to 533.9

seconds when using $\theta = 0.75$ and $\mu = 6$ (a 25%) and the response time is reduced from 4.6 to 2.8 seconds (a 35%). The reason of this improvement is that while the learning process proceeds, the case base increases its competence by storing new cases. At the end of the learning process, rarely new cases are added. This directly translates into a smaller number of invocations to the high-level planner as shown in the column *Planner* from Table 1. In this point, the policy π_B^θ has been learned and most of the unexpected situations are conducted by π_θ^B and not by the high-level planner. In the case of OAKPlan, for the problem $70 \times 40 \times 16$, the total time is reduced from 637 to 533.9 seconds when using $\theta = 0.75$ and $\mu = 6$ (a 16%) and the response time is reduced from 3.9 to 2.8 seconds (a 28%). Therefore, in summary, the results in Table 1 demonstrate, on one hand, that the case based policy helps to reduce the response times of CB-PELEA and, on the other hand, that storing partial plans is better than storing only one action or entire plans. Figure 9 reinforces the previous conclusions. It shows the evolution of the learning process of CB-PELEA ($\theta = 0.75$ and $\mu = 6$) for the problem $70 \times 40 \times 16$ across four different dimensions: cost, response time, cases added, and number of invocations to the high-level planner. Figure 9 (a) shows how the cost increases throughout the episodes. At the end of the learning process, the learned case-based policy needs to perform about 10 actions more in order to reach the goals. However, Figure 9 (b) shows how the response time decays inasmuch the competence of the case base increases (Figure 9 (c)), which in turn causes a reduction of the number of invocations of the high-level planner (Figure 9 (d)). In any case, the experiments demonstrate that CB-PELEA drastically improves the response time of OAKPlan and FD-Replan.

Finally, Figure 10 shows the transfer learning capability of our approach in CB-PELEA. Figure 10 (a) shows the transfer between two problems: the source one, which is easier, $\Omega_{past} : 20 \times 10 \times 6$, and the target one $\Omega_{new_1} : 40 \times 30 \times 8$. It shows the evolution of the response time for three learning processes: the solid red line shows response time during the learning process of Ω_{new_1} from scratch, while the dashed green line shows the response time when learning the problem Ω_{past}, also from scratch. The dashed blue line is the response time when learning the problem Ω_{new_1}, by reusing the case base policy learned previously for Ω_{past}, i.e, using the case base B_{past} learned in the problem $20 \times 10 \times 6$ to learn a new case base B_{new_1} for the problem $30 \times 20 \times 7$. Figure 10 (b) shows the same information but to solve the problem $\Omega_{new_2} : 50 \times 40 \times 9$ by reusing the case base policy learned for $\Omega_{past} : 20 \times 10 \times 6$.

Both figures show the benefits to the jump start, or difference between the initial point with and without transfer learning [47], that is decreased from 1.2 seconds to around 0.8 seconds in Figure 10 (a). However, the benefit of the jump start in Figure 10 (b) is less pronounced since the new problem to solve, Ω_{new_2}, is more different to the problem Ω_{past} than Ω_{new_1}. Therefore, the usefulness of reusing a policy depends on the similarity of the new problem Ω_{new} and the past problem Ω_{past}. Finally, in both cases, the reuse also produce a light benefit in the final

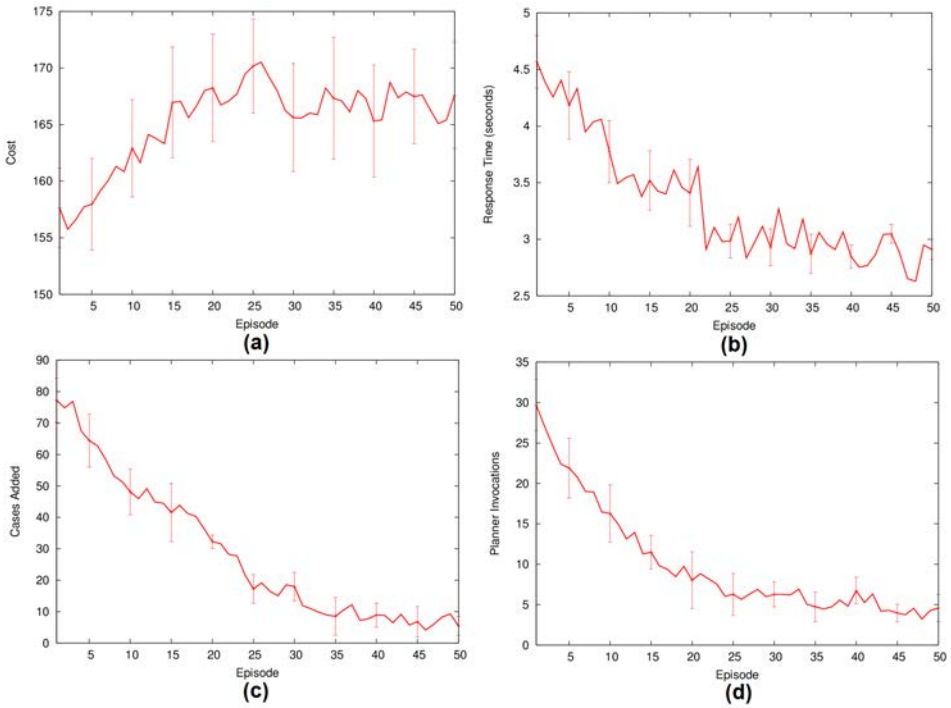

Figure 9. Evolution of the learning process for the problem $70 \times 40 \times 16$ across four different dimensions: (a) Cost, (b) Response time, (c) Cases Added, and (d) Number of invocations to the high-level planner.

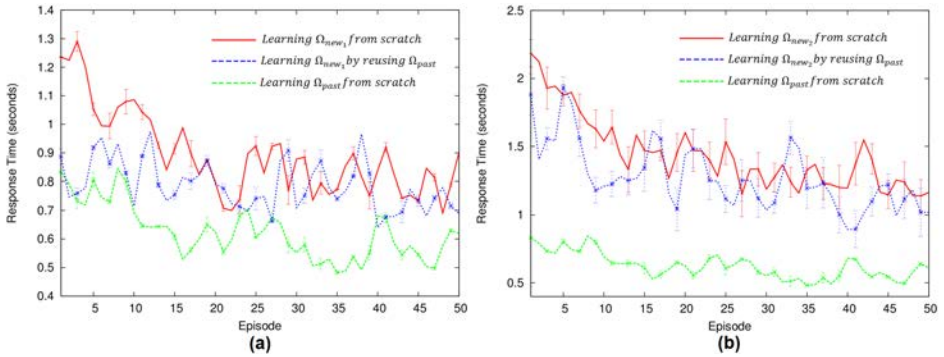

Figure 10. (a) Transfer learning from the problem Ω_{past} : $20 \times 10 \times 6$ to the problem Ω_{new_1} : $40 \times 30 \times 8$, and (b) transfer learning from the problem Ω_{past} : $20 \times 10 \times 6$ to the problem Ω_{new_2} : $50 \times 40 \times 9$.

asymptotic performance.

7.3. *Gold-Miner Domain Results*

This domain is a variant of the domain designed for the learning track of IPC-6 (2008) and it was inspired on the gold extraction process in a mine. In this domain, there is a set of robots in a mine that can reach different locations that contains gold to extract it. The mine is defined as a grid with each cell being either hard or soft rock. The robot can use two different tools to destroy rocks: bombs and a laser cannon. The laser cannon can shoot through both hard and soft rock, whereas the bomb can only penetrate soft rock. However, the laser cannon also will destroy the gold if used to uncover the gold location. The bomb does not destroy the gold. For this domain, we have designed an error model based on four failures which have been encoded into PPDDL: (i) there is a general error with probability of 0.05 which prevent the execution of any action (i.e., the action is not executed), (ii) there is a bomb detonation error with probability of 0.1 which prevents that a soft rock is destroyed when a bomb detonates, (iii) there is a laser manipulation error with probability 0.1 which makes the robot's laser falls to the ground without destroying the hard rocks; and (iv) there is a navigation error, i.e, the robot navigates to an unexpected cell when is navigating with probability 0.1.

Table 2 shows the results for three different problems increasing in complexity using OAKPlan [44], FD-Replan, and CB-PELEA using different values for the parameters θ and μ. The results in Table 2 are presented across the seven dimensions previously described in Section 7.1: *Cost, Time, Response, Planner, Discards, Cases and Size*. The values for the parameters θ and μ are computed heuristically as described in Section 7.1. In this case, $\theta = 0.95$ and $\mu = 3$ for the simplest problem, $\mu = 4$ for the medium complexity problem, and $\mu = 5$ for the most complex problem. The problem size (*Problem Size*) is expressed as $\#rows\ grid\ cell \times \#columns\ grid\ cell \times \#goals$.

Results in Table 2 also shows the three main conclusions described in the previous section for Rovers Domain. First, if θ is equal to 0, the cost of solving the problems is similar both in FD-Replan and CB-PELEA; but if $\theta = 0.95$, this cost increases. For instance, for the problem $7 \times 7 \times 5$, the cost of FD-Replan is 160.1, but the cost when $\theta = 0.95$ is 171.2 for $\mu = 1$, 173.4 for $\mu = 5$ and 172.3 for μ equal to the entire plan. This is because the use of $\theta > 0$ may cause the retrieval of plans that do not perfectly fits to the current situation, and therefore, a greater number of actions are required to solve the problems. In any case, although there is an increament in the cost when $\theta = 0.95$, the problems are successfully solved. Additionally, the cost of solving the problem by OAKPlan is also lower than the cost of CB-PELEA. For instance, for the problem $7 \times 7 \times 5$, the cost of OAKPlan is 162.1.

Secondly, the CB-PELEA configurations with $\theta = 0.95$ take less time to solve the problem than those with $\theta = 0$. Therefore, the smaller the case base, the faster CB-PELEA solving the problem. For instance, for the problem $7 \times 7 \times 5$, the time for $\theta = 0$ and $\mu = 1$ is 189.6 and it is reduced to 177.2 seconds when $\theta = 0.95$ and

Problem	Approach		Cost	Time	Response	Planner	Discards	Cases	Size
5 × 5 × 3	OAKPlan		75(3.8)	21.2(3.1)	0.3(0.08)	3(1.1)	17.2(2.3)	65.3(1.3)	10.7(0.4)
	FD-Replan		**72.3**(3.8)	28.2(3.6)	0.5(0.1)	13(2.8)	-	-	-
	CB-PELEA	θ=0 μ=1	74.2(4.3)	24.1(2.6)	0.35(0.07)	0(0)	**0**	68(6.9)	0.8(0.2)
		θ=0 μ=3	75.1(3.2)	19.4(4.2)	0.25(0.04)	2(0.8)	1.2(0.2)	70.4(5.1)	0.8(0.3)
		θ=0 μ=full	73.7(5.8)	18.9(3.7)	0.3(0.06)	1(0.5)	16.4(3.4)	74.1(6.5)	6.1(0.3)
		θ=.95 μ=1	80.2(6.1)	16.2(4.6)	0.22(0.01)	3(0.8)	**0**	**43.1**(4.5)	**0.5**(0.1)
		θ=.95 μ=3	83.1(4.8)	**12.2**(2.3)	**0.14**(0.03)	2(0.2)	0.9(0.8)	52.1(3.7)	0.6(0.2)
		θ=.95 μ=full	82.4(3.4)	15.8(3.5)	0.19(0.03)	1(0.2)	14.2(3.5)	**44.1**(6.4)	3.2(0.2)
6 × 6 × 4	OAKPlan		134.2(5.2)	92.1(4.1)	0.9(0.1)	4(3.2)	38.3(4.3)	138.4(7.2)	29.5(1.2)
	FD-Replan		133.2(6.1)	106.5(7.9)	1.2(0.1)	19.6(3.4)	-	-	-
	CB-PELEA	θ=0 μ=1	135.5(5.4)	99.6(4.3)	0.9(0.01)	0.9(0.7)	**0**	144.3(13.2)	4.3(0.8)
		θ=0 μ=4	**132.1**(5.3)	70.2(6.2)	0.6(0.05)	1.4(0.9)	1.3(0.7)	158.1(10.4)	4.7(1.1)
		θ=0 μ=full	135.2(3.4)	87.3(3.3)	0.8(0.08)	2.1(0.8)	37(9.7)	139.4(14.3)	19.4(0.9)
		θ=.95 μ=1	145.4(7.4)	72.3(5.3)	0.5(0.04)	1.8(0.6)	**0**	86.4(8.3)	**2.5**(0.7)
		θ=.95 μ=4	144.1(4.3)	**48.8**(3.9)	**0.3**(0.02)	1.2(1.1)	1.8(0.6)	**92.3**(8.2)	2.7(0.8)
		θ=.95 μ=full	139.4(5.6)	65.3(4.7)	0.47(0.05)	2.1(0.7)	44(12.6)	95.3(4.7)	16.8(0.7)
7 × 7 × 5	OAKPlan		162.1(8.6)	182(13.7)	1.1(0.2)	3(1.3)	50.2(3.4)	204.2(14.7)	64.4(1.7)
	FD-Replan		160.1(6.2)	244.1(11.3)	1.7(0.2)	30(5.2)	-	-	-
	CB-PELEA	θ=0 μ=1	**157.3**(3.2)	189.6(10.7)	1.3(0.1)	2.8(1.4)	**0**	207(15.2)	8.2(0.9)
		θ=0 μ=5	164.4(5.3)	164.4(9.8)	1.0(0.08)	4.2(1.2)	1.4(0.8)	214(19.3)	8.5(1.2)
		θ=0 μ=full	161.2(7.2)	179.3(11.9)	1.2(0.07)	3.1(1.3)	53(4.8)	225(11.1)	29.1(0.7)
		θ=.95 μ=1	171.2(4.1)	177.2(8.7)	0.9(0.02)	5.7(1.2)	**0**	**131.4**(14.7)	**5.2**(0.8)
		θ=.95 μ=5	173.4(6.2)	**97.9**(4.4)	**0.5**(0.03)	3.5(1.1)	1.1(0.7)	142(14.4)	5.7(1.2)
		θ=.95 μ=full	172.3(4.2)	157.8(14.7)	0.8(0.03)	2.1(2.1)	54(9.2)	138(9.4)	22.5(0.6)

Table 2. Results in Goldminer domain for different problem sizes and approaches across seven dimensions. Results for OAKPlan and CB-PELEA are measured after 20 episodes. Standard deviations are in brackets.

$\mu = 1$, the time for $\theta = 0$ and $\mu = 5$ is reduced from 164.4 to 97.9 seconds when $\theta = 0.95$ and $\mu = 5$ and, finally, the time for $\theta = 0$ and μ equal to the entire plan is reduced from 179.3 to 157.8 seconds when $\theta = 0.95$ and μ equal to the entire plan. As in the previous section, OAKPlan requires a longer time to solve the problems than most of the configurations of CB-PELEA since it presents a more complex retention and retrieval processes.

Among the configurations where $\theta = 0.95$, the best times are those where μ is computed heuristically. On one hand, for the problem $7 \times 7 \times 5$, the total time for $\theta = 0.95$ and $\mu = 1$ is reduced from 177.2 to 97.9 seconds, and the response time is reduced from 0.9 to 0.5 seconds when $\theta = 0.95$ and $\mu = 5$. On the other hand, the total time for $\theta = 0.95$ and μ equal to the full plan is reduced from 157.8 to 97.9 seconds, and the response time is reduced from 0.8 to 0.5 seconds when $\theta = 0.95$ and $\mu = 5$. Additionally, when we retrieve and adapt entire plans, many of the adapted actions are discarded because an unexpected situation happens before the plan is entirely executed. For instance, for the problem $7 \times 7 \times 5$, when $\theta = 0.95$ and μ equal to the full plan, it is adapted on average 54 actions of the retrieved plan that will be discarded. Instead, when $\theta = 0.95$ and $\mu = 5$, the number of discarded

actions per retrieval is reduced to 1.1. Additionally, storing full plans leads to an unnecesary memory consumption. For the problem $7 \times 7 \times 5$, when $\theta = 0.95$ and μ equal to the full plan, the size of the case base is 12.5 megabytes. Instead, when $\theta = 0.95$ and $\mu = 5$ the size of the case base is 5.7 megabytes. Instead, OAKPlan requires the largest case base, since it stores more information in each case. For the problem $7 \times 7 \times 5$, OAKPlan requires a case base of 64.4 megabytes.

The third and last aspect for consideration in Table 2 is that the use of CB-PELEA clearly outperforms the response time of OAKPlan and FD-Replan. In the case of FD-Replan, for the problem $7 \times 7 \times 5$, the total time is reduced from 244.1 to 97.9 seconds when using $\theta = 0.95$ and $\mu = 5$ and the response time is reduced from 1.7 to 0.5 seconds. As in the case of the rovers domain, at the end of the learning process, rarely new cases are added to the case base. This translates into a smaller number of invocations to the high-level planner as shown in the column *Planner* from Table 2. In this point, most of the unexpected situations are conducted by the learned case-based policy π_θ^B and not by the high-level planner. In the case of OAKPlan, for the problem $7 \times 7 \times 5$, the total time is reduced from 182 to 97.9 seconds when using $\theta = 0.95$ and $\mu = 5$ and the response time is reduced from 1.1 to 0.5 seconds. Therefore, in summary, the results in Table 2 also demonstrate that, on one hand, that the case based policy helps to reduce the response times of CB-PELEA and, on the other hand, that storing partial plans is better than storing only one action or the entire plans.

Figure 11 shows the evolution of the learning process of CB-PELEA ($\theta = 0.95$ and $\mu = 5$) for the problem $7 \times 7 \times 5$ across four different dimensions: cost, response time, cases added, and number of invocations to the high-level planner. It shows how the cost increases throughout the episodes. At the end of the learning process, the learned case-based policy needs to perform about 10 actions more in order to reach the goals. However, Figure 11 (b) shows how the response time is reduced inasmuch the case base increases its competence (Figure 11 (c)), and the number of invocations to the high-level planner decreases (Figure 11 (d)).

Finally, Figure 12 shows the results for the transfer learning experiments. Figure 12 (a) shows the transfer between $\Omega_{past} : 5 \times 5 \times 3$ and the problem $\Omega_{new_1} : 6 \times 6 \times 4$. The figure shows the evolution of the response time for three learning process: learning Ω_{past} from scratch (green dashed line), learning task Ω_{new_1} from scratch (solid red line), and learning task Ω_{new_1} through the transfer of previously learned policy $\pi_{B_{past}}$. Besides, Figure 12 (b) shows the same information in transfer learning between $\Omega_{past} : 6 \times 6 \times 4$ and $\Omega_{new_2} : 7 \times 7 \times 5$. The same conclusions as in the Rovers domain arises: benefits to the jumpstart, that reduces the initial response time in Figure 12 (a) from around 0.9 seconds down to 0.4 seconds. In Figure 12 (b) this benefit is not so obvious because the bigger differences between Ω_{past} and Ω_{new_2} than between Ω_{past} and Ω_{new_1}. In this case, no difference in the asymptotic performance seems to be found.

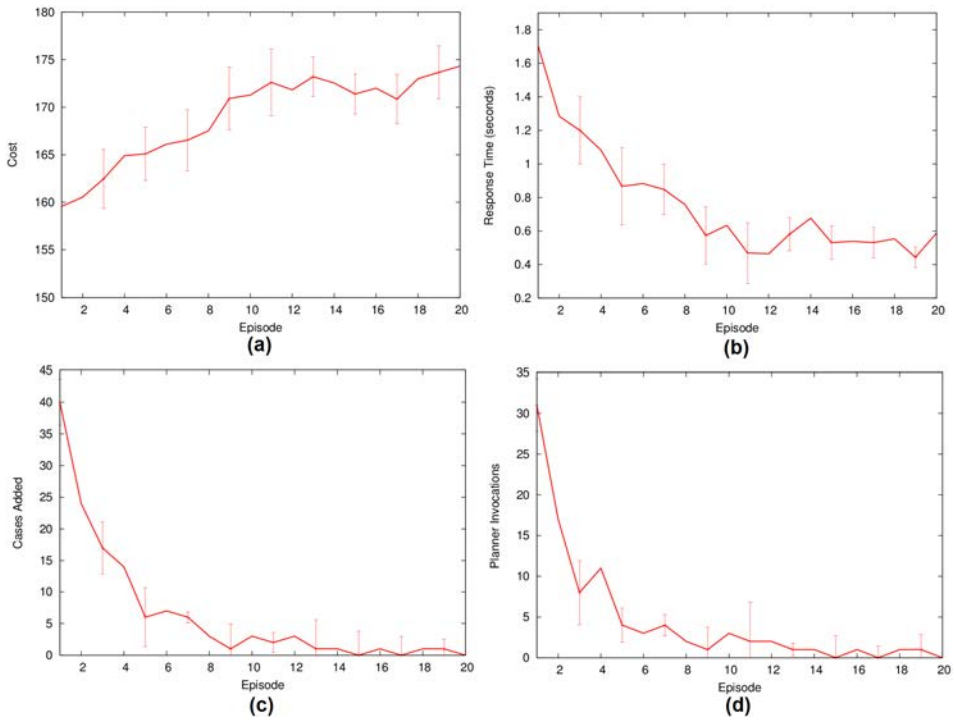

Figure 11. Evolution of the learning process for the problem $7 \times 7 \times 5$ across four different dimensions: (a) Cost, (b) Response time, (c) Cases Added, and (d) Number of invocations to the high-level planner.

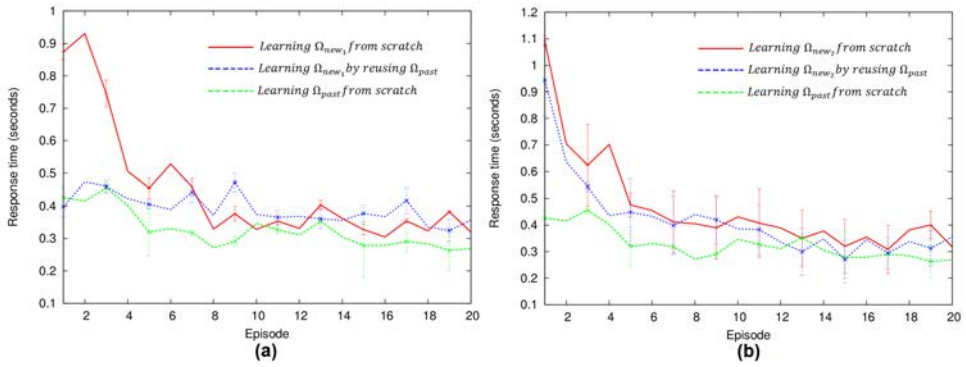

Figure 12. (a) Transfer learning from the problem $\Omega_{past} : 5 \times 5 \times 3$ to the problem $\Omega_{new_1} : 6 \times 6 \times 4$ and (b) transfer learning from the problem $\Omega_{past} : 5 \times 5 \times 3$ to the problem $\Omega_{new_2} : 7 \times 7 \times 5$.

7.4. *Barman Domain Results*

Barman domain was designed for IPC-7 (2011). In this domain, there is a robot barman that manipulates drink dispensers, glasses and a shaker. The goal is to

find a plan of the robot's actions that serves a desired set of drinks. In this case, we have encoded two failures into PPDDL: (i) there is an error with probability 0.5 which prevent the execution of any action and (ii) there is a pour shot error with probability 0.3 which causes the drink spills and do not fill the container to the corresponding level. Table 3 shows the results for three different problems increasing in complexity using OAKPlan, FD-Replan and CB-PELEA. The problem size is expressed as $\#cocktails \times \#ingredients \times \#goals$. Regarding CB-PELEA, in contrast to previous sections, Table 3 only shows the configuration having the best response time. In this configuration, both θ and μ are computed heuristically. The parameter θ is set to $\theta = 0.7$ and μ is computed for each problem ($\mu = 6$, $\mu = 7$ and $\mu = 8$ respectively).

Problem	Approach	Cost	Time	Response	Planner	Discards	Cases	Size
$3 \times 5 \times 7$	OAKPlan	54.3(5.8)	38.1(3.5)	0.7(0.04)	8(3.2)	21.5(2.4)	989.3(34.2)	33.5(1.7)
	FD-Replan	**51.2**(3.4)	56.1(4.6)	1.1(0.1)	18(3.6)	-	-	-
	CB-PELEA	61.8(5.8)	**18.3**(2.3)	**0.3**(0.01)	**6**(1.2)	**1.4**(0.5)	**578.3**(23.8)	**6.5**(1.8)
$6 \times 10 \times 14$	OAKPlan	121.3(2.5)	157.7(6.3)	1.3(0.2)	10(3.1)	34.2(3.7)	1539.7(45.4)	53.6(3.7)
	FD-Replan	**118.2**(3.4)	212.3(12.5)	1.8(0.3)	35(2.1)	-	-	-
	CB-PELEA	131.4(3.8)	**78.6**(5.5)	**0.6**(0.2)	**7**(2.4)	**2.1**(0.5)	**923.7**(35.7)	**10.3**(2.3)
$10 \times 20 \times 28$	OAKPlan	195.2(8.8)	351.2(9.4)	1.8(0.3)	12(3.4)	47.3(5.5)	2184.3(52.4)	63.5(1.7)
	FD-Replan	**188.7**(4.8)	394.8(14.3)	2.1(0.4)	42(5.1)	-	-	-
	CB-PELEA	203.1(5.4)	**223.3**(8.3)	**1.1**(0.4)	**6**(2.6)	**2.4**(1.5)	**1339.7**(52.7)	**17.3**(2.7)

Table 3. Results in Barman domain for different problem sizes and approaches across seven dimensions. Results for OAKPlan and CB-PELEA are measured after 50 episodes. Standard deviations are in brackets.

Table 3 provides the same conclusions as the previous experiments. On one hand, the use of $\theta > 0$ causes the retrieval plans do not perfectly fits to the current situation and, hence, a greater number of actions are required to solve the problem. On the other hand, the response time of CB-PELEA clearly outperforms the response time of FD-Replan and OAKPlan. This is so due to the reuse of cases in similar situations, and to the recovery and adaptation of partial plans instead of entire plans. Finally, Figure 13 shows the result for the transfer between $\Omega_{past} : 3 \times 5 \times 7$ and $\Omega_{new_1} : 6 \times 10 \times 14$ (Figure 13 (a)), and the transfer between $\Omega_{past} : 3 \times 5 \times 7$ and $\Omega_{new_2} : 10 \times 20 \times 28$. Both experiments show the benefits of transfer learning: the initial response time is reduced. However, this benefit is nearly unnoticeable in Figure 13 (b) due to the high difference between Ω_{past} and Ω_{new_2}.

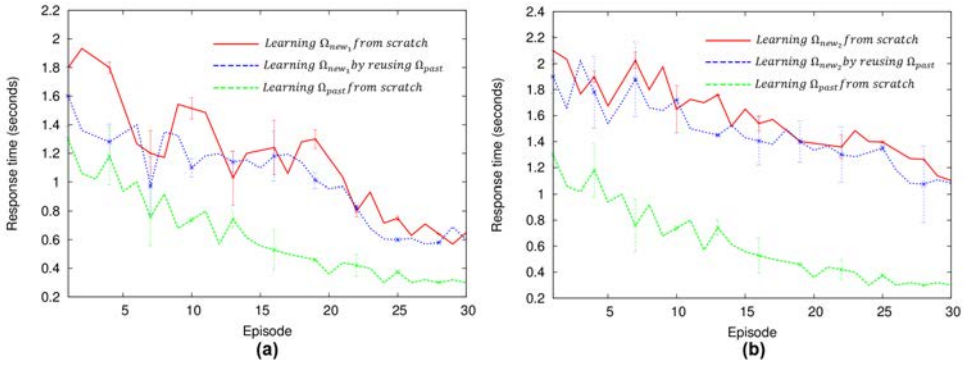

Figure 13. (a) Transfer learning from the problem Ω_{past} : $3\times5\times7$ to the problem Ω_{new_1} : $6\times10\times14$ and (b) transfer learning from the problem Ω_{past} : $3 \times 5 \times 7$ to the problem Ω_{new_2} : $10 \times 20 \times 28$.

8. Related Work

Automated Planning and Case-Based Reasoning techniques have been used in the literature in different ways. Learning by Case-Base Reasoning applied to Automated Planning is known as *Case-Based Planning* (CBP), or *planning by reuse* [7].

We consider there are two basic categories on CBP depending on the information stored: transformational category and derivational category. In the first one, the information stored consists of solutions, which for CBP corresponds to plans. Then, these stored plans are reused by adapting them to the new situations. SPA [25] (based on NONLIN [46]), Priar [30] (also based on NONLIN), CHEF [24] or MAYOR [13] are systems inside this category and they need to store the entire plans. In these approaches, the adaptation of entire plans may be very expensive given that the transformation of the retrieved plan given the current situation is a source of additional hardness. It may make the plan adaptation even more expensive than the traditional plan generation [35]. This is aggravated when we have large plans as the considered in Section 7. However, the results in Section 7 demonstrate that the adaptation of entire plans may no be necessary when considering stochastic domains because most of the adapted actions end up by being discarded. In contrast, the cases in our approach only store partial plans which results in a fast adaptation of the retrieved plans and in a reduction of the memory consumption. Intuitively, we might think that existing approaches can be easily adapted to store partial plans instead of entire plans, but most of these approaches are closely tied to the use of entire plans, and hardly they could be modified to use partial plans. Such is the case of OAKPlan [44] which tries to reuse as much of the stored plan as possible to reach the goals of the new problem.

FAR-OFF [48] is another case-based planner inside this transformational category. It is the first case-based planner adopting a heuristic approach to the retrieval and adaptation phases. However, this planner forces to reuse a previous solution, even though the system is capable of heuristically choosing between reusing a case or

generating a solution from scratch if no suitable case is found in the case base. This fact may lower the quality of the reuse plans. In contrast, if our approach does not find a similar case in the case base, it generates a plan from scratch (Section 4.3). Additionally, FAR-OFF and other systems like MAYOR, CHEF or OAKPlan does not focus on how to learn the cases in the case base. They simply accumulate in the case base all the new cases produced, without implementing any mechanism to decide whether it is advisable to store a new case or not. In contrast, CB-PELEA attempts to solve the case acquisition in an on-line manner using the parent-child analogy described in Section 4.3. In this case, the threshold θ is used to decide whether to include a new case in the base case.

Additionally, most of these techniques are modeled as a single-shot process, i.e. a single loop in the CBR cycle solves a problem. They work in a off-line manner retrieving a single plan and adapting it to solve a new problem. Most of these planners do not include the monitoring of the executed plan (potentially having to retrieve new cases and to adapt it along the way). They are not conceived to work in an on-line manner in stochastic and complex environments where it is needed more than one retrieve/adaptation process and where the response time is critical. In contrast, we propose an on-line system that plans and executes plans in real time, and which uses a case-based policy with partial plan to produce timely responses. As an exception, Darmok architecture [37] implements an on-line case-base planner (OLCBP) to generate a case library by learning from demonstration on WARGUS (an open source clone of WarCraft II video-game). This architecture uses a planning, execution, learning and monitoring loop, which is quite similar to the one in our approach. As our work each case store a partial plan but, in constrast with our work, it also stores a set of subgoals that the execution of the partial plan in the case must be fulfilled. In this way, the case in the Darmok architecture can be seen as macro-operators in planning, each one fulfilling a subgoal in order to fulfill the general goal. This implies an additional effort in analyzing the plan trace identifying partial plans with its corresponding sub-goal, each one with a different length. In addition, this analysis is hand-made by a human expert. Instead, the partial plans in our cases have a fixed size, do not are used to fulfill subgoals, but to fulfill in the future the global goals stored in the case. Additionally, our approach learns from the trace generated by a planner, and Darmok learns from the trace generated by a human (whose decisions and annotations are based on their own feelings). In our case, a human expert is not used in any way.

In the derivational category, CBP systems include the derivational trace in addition to the plan itself. This trace is a sequence of computational steps a planner followed to generate a plan. They focus on the use of this trace to guide the search for a new solution plan rather than in adapting only the plan itself as in the transformational category. Therefore, it can be considered that we address a different kind of problem, because we do not try to modify the exploration process conducted by the planner. The PRODIGY / ANALOGY [50] system is the first case-based planner that executes the case-based cycle using derivational traces. Lines of reasoning are

transferred to the new problem, reproducing/replaying the steps stored in the retrieved cases. However, in contrast with our approach, ANALOGY which constantly consults the case base to retrieve cases that contain similar planning situations and are used to constrain the search space of the planner, while our consults are delayed to the value of μ or when a unexpected state is detected. De la Rosa et al. [11] store partial plans but from the point of view of a specific object. Each case represents the plan from the perspective of an object, and use it to recommend nodes in the search process. However, this object-centered case representation loses information on the specific relations among objects. Our case representation stores partial plans in which the relations among objects remain.

Finally, it is important to note that the modification of the search process requires the modification of the planner itself (i.e., it requires to modify the source code). Therefore, the implementation of the ideas proposed by the derivational case-based planners in new state-of-the-art planners such as Fast Downward [26] or Mercury [31], is not straightforward. In contrast, our approach does not require the modification of the source code of the generative planner. In this paper, we have used Fast Downward, but it can be easily replaced by any other planner as the rest of the system is independent on this choice. In this way, we can easily take advantage of the new planners as soon as they appear.

From a different perspective of the approaches in the transformational and derivational categories, CBR has also been applied to hierarchical planning such as in HTN-MAKER [28]. This planning system learns new methods for decomposing hierarchical tasks. It tries to reduce the effort of designing the domain knowledge for controlling the search, which in the case of hierarchical planning is written within the action model. In our approach, the action model is fixed and we use the cases learned to support the planning process.

More recently, policy-learning approaches have been successfully applied to the control of single agents acting under uncertain environments in specific problems. For instance, the work by [18] consists in managing the loading of multiple independent battery cells in order to maximize their lifetime and reduce switching between batteries. Another one [19] consists in controlling underwater vehicles which exploratory mission. In particular, in that paper, CBR is applied to the problem of tracking the outer edge of 2D biological features, such as the surfaces of harmful algal blooms. However, traditional approaches learn these policies in an off-line way [18,19]: (i) sampling many instances of the stochastic problem, each instance being a challenging temporal and metric planning problem; (ii) solving each instance using a high-performing planner; and (iii) applying a classifier to learn the policy. Learning the case-based policy in an on-line way provides two main benefits over the off-line learning [19]: (i) the case-based policy is built progressively using the experience gathered in real executions, and (ii) avoids storing rare cases in the case base which are not used later in real executions. Durán et al. [22] also learn a plan-based policy. Specifically, large set of plans generated by the planner on different

planning problems is stored in terms of state-goal action tuples. However, Durán et al. [22] use the policy to improve the search process in planning, and we use it to directly provide actions to the environment.

Other approaches based on BDI (Belief-Desire-Intention) are similar to ours since they also build a *plan library* [42,45]. However, these approaches are intrinsically tied with Hierarchical planners, which require to incorporate additional expert knowledge into the domain description in order to solve the tasks (e.g., for the decomposition of the tasks in subtasks). Nevertheless, this expert knowledge is not always available. In contrast, our approach use classical planning which does not require such expert knowledge. Additionally, these approaches store all plans and entire plans in this library which is already impossible for real world problems and searching for the applicable one is another impossible tasks. In contrast, we store partial plans instead of entire plans; and, on the other hand, we prevent to store similar cases in the case base, i.e., similar plans. Last, these approaches do not perform the CBR cycle as our approach do.

Finally, non-deterministic or Probabilistic planners [27] are extremely interesting approaches to deal with real world problems. However, such approaches applied to real-world problems have to deal with other significant problem: in most cases the transition model is unknown. This makes many probabilistic planners rely on Reinforcement Learning (RL) techniques. For instance, FPG gradient borrows from Policy-Gradient RL and, hence, it does not need to estimate planning state-action values [3]. Instead, UCT-based PROST planner needs to estimate state-action values for each node of the search tree [1]. In all these approaches, the state space must be explored (in most cases randomly) and this exploration can lead to catastrophic consequences or dead-ends many times before the system learns to avoid them. Other interesting approaches are based on using Domain Control Knowledge (e.g., macros [15], reactive policies [53]) for speeding-up plan generation. However, we consider a more extent discussion on all these approaches is beyond the scope of this paper. This paper restricts its attention on the application of deterministic planning and CBP to solve probabilistic problems.

9. Conclusions

This paper describes a learning algorithm to learn a case-based policy using on-line plan executions. This component has been integrated in the robot control architecture PELEA. The Rovers, Goldminer and Barman domains has been used to illustrate the performance of the proposed learning approach. Next we summarize the main conclusions found in this paper:

(i) Fast response time for unexpected situations. The experiments in Section 7 demonstrate that the case-based policy learned is able to produce faster responses than planning from scratch when an unexpected situation happens. For all the domains and the configurations proposed, CB-PELEA outperforms the response time of FD-Replan and OAKPlan [44]. Taking into account the largest prob-

lems for each domain, the response time of CB-PELEA is 40% faster than the response time of OAKPlan, and 53% faster than the response time of FD-Replan. This is particularly interesting in realistic environments where there are more and more unexpected situations, and where timely responses in such situations is really essential.

(ii) Novel mechanisms in the reduction of the size of the case-base for real world problems. Although the idea of storing plans is not new, in this paper we introduce two novel mechanisms to reduce the size of the case base in order to tackle real world problems. On one hand, we store partial plans instead of entire plans; and, on the other hand, we prevent to store similar cases in the case base. Additionally, we have integrated successfully these two mechanisms in a CBR cycle.

(iii) It is better to store partial plans than entire plans. The longer the retrieved plan, the higher the time required to adapt it. Furthermore, the experiments in Section 7 demonstrate that in stochastic environments most of the adapted actions of the entire plan have to be discarded because an unexpected situation happens before the plan is fully executed. Therefore, the adaptation of entire plans in such environments leads to an unnecessary time and memory consumption. Storing partial plans reduces the effort of adaptation, the number of discarded adapted actions, and the size of the case base. Section 7.1 gives a guideline to heuristically compute the size μ of the partial plans stored, that can be done off-line. Obviously, we in no way state that the plan adaptation proposed in this paper is better than the proposed by LPG-adapt. However, in the context of on-line execution of plans in stochastic environments, the experiments in Section 7 demonstrate that it is not worth using techniques such as LPG-adapt for the adaptation of the entire plan. In this context, we have to adapt as fast as possible the number of actions that are believed to be performed before a new unexpected situation occurs.

(iv) Increment in the cost of the solutions. One of the proposed mechanisms to reduce the size of the case base is based on the fact that similar situations are expected to be solved with similar plans. In large environments with a huge number of states, it is impossible to store a plan for each of these states. Therefore, the parameter θ is used for deciding whether to store a new case in the case base or not. However, although it is expected that similar plans in similar states tend to produce similar effects, this could lead to an increment in the cost of the solutions (i.e., it is required a higher number of actions to solve the problem). In any way, the experiments in Section 7 demonstrate that, using a correct value of the parameter θ, the cost is not greatly affected and the size of the case base is significantly reduced (with the consequent reduction of the response time). Section 7.1 provides a mechanism to heuristically compute the value of θ.

(v) On-line acquisition of new cases. Most of the work on case-based planning does not focuses on how to learn the cases in the case base. So for instance, systems like MAYOR [13] or CHEF [24] use a predefined case-base. Other systems [11,18,19] uses a previous off-line training phase before the on-line execution. In our case, the

case-based policy is built progressively in an on-line manner using the parent-child analogy described in Section 4.3.

(vi) Ability to easily take advantage of improvements in generative planners. Derivational CBP systems are based on the modification of the search process to achieve the solutions, and these modifications are closely linked to the planning system used. Thus, for example, the transferring of the modifications that implements CABALA [11] to Fast Downward [26] requires knowing the internal functioning of Fast Downward and then modify its source code. In contrast, in this paper we use Fast Downward for planning from scratch, but it can be easily replaced by any other planner as the rest of the system is independent on this choice. In this way, in the future we could replace Fast Downward with another better planner and, thus, get better response times.

(vii) Transfer knowledge from one problem to a similar problem. Transfer learning experiments show that the case-based policy learned in a particular problem can be reused to solve a similar problem. In this way, the number of re-planning situations conducted by the high-level planner are reduced from the beginning, reducing, at the same time, the response time from the early steps of the learning process. However, the greater the differences between the problems, the smaller the benefit of transferring knowledge.

(viii) Low number of CBP systems for on-line execution. In the literature there exists a low number of CBP systems for monitoring the execution of a plan [37]. Most of them work in a off-line manner retrieving a single plan and adapting it to solve a new problem. However, on-line execution opens new challenges. Monitoring the plan execution using CBP implies that all the phases of CBR (retention, retrieval and reuse) should be performed in real-time and more than once and, hence, these steps should be as fast as possible in order to ensure good response times. In this paper, we have addressed these challenges and we have proposed a CBP system for on-line monitoring and execution able to reduce considerably the response times.

As future work, we consider implementing other distance functions to reduce the size of the case base, and to improve the reuse phase in order to obtain better quality plans. We also consider an important part of the future work implementing a mechanism to avoid the visit to dead-end states. Finally, we also consider the deployment of the architecture in real robot domains, and exhaustive testing processes in real environments.

Acknowledgment

This paper has been partially supported by the Spanish Ministerio de Economía y Competitividad TIN2015-65686-C5-1-R and the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO).

Bibliography

1. Prost: Probabilistic planning based on uct. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, Jun 2012.
2. Agnar Aamodt and Enric Plaza. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, 7(1):39–59, 1994.
3. Douglas Aberdeen. Policy-gradient methods for planning. In *Advances in Neural Information Processing Systems*, pages 9–16, 2006.
4. Mitchell Ai-Chang, John L. Bresina, Leonard Charest, Adam Chase, Jennifer Chengjung Hsu, Ari K. Jónsson, Bob Kanefsky, Paul H. Morris, Kanna Rajan, Jeffrey Yglesias, Brian G. Chafin, William C. Dias, and Pierre F. Maldague. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.
5. Eyal Amir and Pedrito Maynard-reid. Logic-based subsumption architecture. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 147–152, 1999.
6. Jim Blythe and W. Scott Reilly. Integrating reactive and deliberative planning in a household robot. In *In Proceedings of the aaai Fall Symposium on Instantiating Real-World Agents*, pages 6–13, 1993.
7. Daniel Borrajo, Anna Roubíčková, and Ivan Serina. Progress in case-based planning. *ACM Computing Surveys*, 47(2):35:1–35:39, January 2015.
8. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(10), 1986.
9. Rodney A. Brooks. Integrated systems based on behaviors. *Stanford University*, 2:46–50, 1991.
10. Marc de la Asunción, Luis A. Castillo, Juan Fernández-Olivares, Óscar García-Pérez, Antonio González, and Francisco Palao. SIADEX: an interactive knowledge-based planner for decision support in forest fire fighting. *AI Communications*, 18(4):257–268, 2005.
11. Tomás de la Rosa, Angel García-Olaya, and Daniel Borrajo. A case-based approach to heuristic planning. *Applied Intelligence*, 39(1):184–201, 2013.
12. Christian Dornhege. *Task Planning for High-Level Robot Control*. PhD thesis, University of Freiburg, 2015.
13. Mark Fasciano. Everyday-world plan use. Technical report, University of Chicago, 1996.
14. Fernando Fernández, Javier García, and Manuela M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, pages 866–871, 2010.
15. Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.
16. M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *Proceedings of International Conference on AI Planning and Scheduling (ICAPS)*. AAAI Press, 2006.
17. Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.*, 20(1):61–124, December 2003.
18. Maria Fox, Derek Long, and Daniele Magazzeni. Automatic construction of efficient multiple battery usage policies. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011.
19. Maria Fox, Derek Long, and Daniele Magazzeni. Plan-based policy-learning for au-

tonomous feature tracking. In *Proceedings of the Twenty-Second International Conference on AutomatedPlanning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 2012.
20. Javier García and Fernando Fernández. Safe exploration of the state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research (JAIR)*, 45:515–564, 2012.
21. Javier Garcia, Jose E. Florez, Alvaro Torralba, Daniel Borrajo, Carlos Linares Lopez, Angel Garcia-Olaya, and Juan Saenz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operational Research (EJOR)*, 227:216–226, 2013.
22. Rocío García-Durán, Fernando Fernández, and Daniel Borrajo. A prototype-based method for classification with time constraints: A case study on automated planning. *Pattern Analysis and Applications Journal*, 15(3):261–277, 2012.
23. Javier García, José E. Florez, Álvaro Torralba, Daniel Borrajo, Carlos Linares López, Ángel García-Olaya, and Juan Sáenz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operational Research*, 227(1):216–226, 2013.
24. Kristian J. Hammond. Case based planning: A framework for planning from experience. *Cognitive Science*, 14(3):385–443, 1990.
25. Steve Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:128–137, 1995.
26. Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
27. Andreas Hertle, Christian Dornhege, Thomas Keller, Robert Mattmüller, Manuela Ortlieb, and Bernhard Nebel. An experimental comparison of classical, FOND and probabilistic planning. In *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*, pages 297–308, 2014.
28. Chad Hogg, Hector Muñoz Avila, and Ugur Kuter. HTN-MAKER: Learning htns with minimal additional knowledge engineering required. In *Proceedings of the 20th AAAI Conference*. AAAI Press, 2008.
29. Leslie Pack Kaelbling and Tomas Lozano-Perez. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*, 2011. Finalist, Best Manipulation Paper Award.
30. Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Journal of Artificial Intelligence*, 55(2):193–258, 1992.
31. M. Katz and J. Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.
32. M. Kirsten, S. Wrobel, and T. Horvath. Distance based approaches to relational learning and clustering. *Dzeroski S, Lavrac N (eds) Relational data mining*, pages 213–232, 2001.
33. Gang Kou, Daji Ergu, Changsheng Lin, and Yang Chen. Pairwise comparison matrix in multiple criteria decision making. *Technological and Economic Development of Economy*, 22(5):738–765, 2016.
34. Moisés Martínez, Fernando Fernández, and Daniel Borrajo. Planning and execution through variable resolution planning. *Robotics and Autonomous Systems*, 83:214–230, 2016.
35. B. Nebel and J. Kohler. Plan reuse versus plan generation: A complexity-theoretic perspective. *Journal of Artificial Intelligence*, 76:427–454, 1995.
36. Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. *Case-Based*

Planning and Execution for Real-Time Strategy Games, pages 164–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
37. Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. On-line case-based planning. *Computational Intelligence*, 26(1):84–119, 2010.
38. Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010.
39. Ezequiel Quintero, Vidal Alcázar, Daniel Borrajo, Juan Fernández-Olivares, Fernando Fernández, Angel García Olaya, César Guzmán, Eva Onaindia, and David Prior. Autonomous mobile robot control and learning with the pelea architecture. In *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*, 2011.
40. Ezequiel Quintero, Ángel García-Olaya, Daniel Borrajo, and Fernando Fernández. Control of autonomous mobile robots with automated planning. *Journal of Physical Agents*, 5(1):3–13, 2011.
41. K. Rajan, C. McGann, F. Py, and H. Thomas. Robust mission planning using deliberative autonomy for autonomous underwater vehicles. In *Proceedings of the Workshop on Robotics in Challenging and Hazardous Environments, ICRA*, Rome, 2007.
42. Sebastian Sardiña, Lavindra de Silva, and Lin Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 1001–1008, 2006.
43. Julia Schmitt, Michael Roth, Rolf Kiefhaber, Florian Kluge, and Theo Ungerer. Concept of a reflex manager to enhance the planner component of an autonomic/organic system. In *Proceedings of the 8th International Conference on Autonomic and Trusted Computing*, ATC'11, pages 19–30, Berlin, Heidelberg, 2011. Springer-Verlag.
44. Ivan Serina. Kernel functions for case-based planning. *Artif. Intell.*, 174(16-17):1369–1406, November 2010.
45. Dhirendra Singh, Sebastian Sardiña, Lin Padgham, and Geoff James. Integrating learning into a BDI agent for environments with changing dynamics. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2525–2530, 2011.
46. Austin Tate. Generating project networks. In R. Reddy, editor, *Proccedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 888–893. William Kaufmann, 1977.
47. Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.
48. F. Tonidandel and M. Rillo. The far-off system: A heuristic search case-based planning. In *Proceedings of International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*. AAAI Press, 2002.
49. Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.
50. Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in Prodigy: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278, 1993.
51. Manuela M. Veloso, Héctor Munoz-Avila, and Ralph Bergmann. General-purpose case-based planning: Methods and systems. *AI Communications*, 9(3):128–137, 1996. Also in Kunstliche Intelligenz, 1:22-28, 1996, in German (with reversed order of the authors).
52. Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated*

Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007, page 352, 2007.

53. Sung Wook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9:683–718, 2008.
54. Häkan L. S. Younes and Michael L. Littman. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. *In Technical Report CMU-CS-04-162*, 2004.
55. Häkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligente Research*, 24:851–887, 2005.